Concatenated Constrained Coding

A New Approach to Efficient Constant-Weight Codes

Immink, K.S.; Weber, J.H.; Nguyen, T. T. ; Cai, Kui

DOI
10.3390/e28010078

Licence
CC BY

Publication date
2026

Document Version
Final published version

Published in
Entropy: international and interdisciplinary journal of entropy and information studies

Citation (APA)
Immink, K. S., Weber, J. H., Nguyen, T. T., & Cai, K. (2026). Concatenated Constrained Coding: A New Approach to Efficient Constant-Weight Codes. *Entropy: international and interdisciplinary journal of entropy and information studies*, *28*(1), Article 78. https://doi.org/10.3390/e28010078

entropy

MDPI

Article

Concatenated Constrained Coding: A New Approach to Efficient Constant-Weight Codes

Kees Schouhamer Immink [1,*], Jos H. Weber [2], Tuan Thanh Nguyen [3] and Kui Cai [3]

[1] Turing Machines Inc., Willemskade 15, 3016 DK Rotterdam, The Netherlands
[2] Department of Applied Mathematics, Delft University of Technology, 2628 CD Delft, The Netherlands; j.h.weber@tudelft.nl
[3] Science, Mathematics and Technology Cluster, Singapore University of Technology and Design (SUTD), 8 Somapah Rd, Singapore 487372, Singapore; tuanthanh_nguyen@sutd.edu.sg (T.T.N.); cai_kui@sutd.edu.sg (K.C.)
* Correspondence: immink@turing-machines.com

Abstract

The design of low-complexity and efficient constrained codes has been a major research item for many years. This paper reports on a versatile method named concatenated constrained codes for designing efficient fixed-length constrained codes with small complexity. A concatenated constrained code comprises two (or more) cooperating constrained codes of low complexity enabling long constrained codes that are not practically feasible with prior art methods. We apply the concatenated coding approach to two case studies, namely the design of constant-weight and low-weight codes. In a binary constant-weight code, each codeword has the same number, w, of 1's, where w is called the weight of a codeword. We specifically focus on the trading between coder complexity and redundancy.

Keywords: balanced code; concatenated constrained code; constant-weight code; constrained code; Knuth's algorithm; low-weight code; m-out-of-n code

1. Introduction

A constrained channel is not capable to transmit all possible signals, and only certain sequences may be allowed [1]. A constrained code implements these constraints by converting arbitrary source data into permitted signals [2–4]. Designing an invertible mapping from arbitrary, unconstrained source sequences into coded, constrained binary sequences is a fundamental challenge. In a conventional block code, the source data are segmented into small, manageable data blocks, which are then translated into sequences of permitted symbols. The resulting codewords satisfy the given constraints, allowing them to be freely cascaded without violating channel constraints. Naturally, larger block sizes enable more efficient encoding into permitted sequences. Numerous implementation strategies have been explored in the literature.

Implementations fall in three main categories: table look-up, replacement techniques, and arithmetic-based implementations. Table look-up is straightforward, but the size of the data blocks is limited by the maximum size of the tables used [5,6]. The replacement technique successively or iteratively removes forbidden subsequences in the source data to obtain a constrained sequence. Enumeration techniques [4,7–12] use integer arithmetic operations to translate source data into constrained codewords and *vice versa* . However, the required silicon area of the look-up table of integer coefficients does not scale linearly with the block size [13], and it may become unacceptably large with mounting word length,

Academic Editors: T. Aaron Gulliver and Chi Wan Sung

Received: 15 December 2025
Revised: 31 December 2025
Accepted: 7 January 2026
Published: 9 January 2026

precluding their use in practical transmission or storage systems [5]. A second important consideration in enumerative encoding is *error propagation*, as a single bit error in the received codeword may corrupt an entire decoded word [14]. This necessitates the use of strong error correction codes, which in turn require additional redundancy.

Concatenated error correction codes [15] are widely used in data transmission and storage systems. Formed by combining two or more codes, typically referred to as the *inner* and *outer codes*, these constructions provide enhanced error performance while maintaining low decoder complexity. Inspired by this principle, we propose a concatenated constrained coding approach, in which two or more low-complexity constrained codes are combined to construct long constrained codes that are infeasible using conventional methods.

In the encoding device, the source data blocks are divided into two segments: a first segment and a second segment. The first segment of the source data is further partitioned into a plurality of small data subwords that are translated into an allowed codeword using several look-up tables whose sets of constrained output words are mutually disjoint. The second segment of source data is encoded using a second constrained code, which determines which look-up tables are applied during the first encoding stage. The final codeword, found by cascading the output words of the look-up tables, is transmitted. A decoder can uniquely restore the source data encoded by the first and second codes by observing the received codeword. We demonstrate that the concatenated constrained code enables the design of longer codewords with less redundancy than conventional methods, while avoiding the need for impractically large look-up tables.

We demonstrate the versatility of the concatenated constrained coding scheme through two case studies: designing long constant-weight and low-weight codes.

Paper structure: We introduce the fundamentals of concatenated constrained codes in Section 2. Section 3 reviews prior work on the properties of constant-weight codes, which sets a baseline for the new constructions. Section 4 presents new constructions of constant-weight codes using concatenated constrained codes. The construction of very long constant-weight codes is explored in Section 5, where we analyze the application of one concatenated constrained code layered on top of another. Applications to low-weight codes are covered in Section 6. Finally, Section 7 concludes the paper.

2. Concatenated Constrained Code, Basics

We describe an encoder that translates binary source data into n-bit binary codewords, denoted by $\boldsymbol{y}$, which satisfy a given constraint $\mathcal{A}$. Let $\mathcal{S}_{\mathcal{A}}$ denote the set of n-bit codewords $\boldsymbol{y}$ that meet constraint $\mathcal{A}$. Since n is typically too large to permit direct generation of codewords via a single look-up table, the codeword $\boldsymbol{y}$ is divided into k equal-sized m-bit constrained subwords $\boldsymbol{y}_t \in \{0,1\}^m$ and $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_k)$ and $n = km$. The parameter m is chosen to be small enough to allow the use of look-up tables for mapping source data to the subwords $\boldsymbol{y}_t$.

2.1. Major Components Overview

We construct K, $K \leq k$, distinct sets of m-bit subwords, denoted by $S_i \subset \{0,1\}^m$, $0 \leq i \leq K-1$, each satisfying a specific constraint $\mathcal{A}_i$, $0 \leq i \leq K-1$. In practice, only $K \leq k$ distinct look-up tables are required as they can be re-used via multiplexing. Each set S_i enables a bijective mapping $f_i : \{0,1\}^{\nu_i} \to S_i$ of $\nu_i \leq m$ source bits into valid subwords, ensuring compatibility with $\mathcal{A}_i$.

To construct an n-bit codeword $\boldsymbol{y}$, we concatenate k m-bit subwords $\boldsymbol{y}_t$, where each $\boldsymbol{y}_t$ is selected from the corresponding subword set S_{c_t}. The selection is guided by a control code $\boldsymbol{c} = (c_1, \ldots, c_k)$, with $c_t \in \{0, \ldots, K-1\}$. The goal is to ensure that the full codeword $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)$ satisfies the overall constraint $\mathcal{A}$.

2.2. Basic Properties

The concatenated constrained coding scheme adheres to the following conditions:

- The sets S_i, $0 \leq i \leq K-1$, must be pairwise disjoint.
- Each codeword $\boldsymbol{y}$ must encode a fixed number of source bits.
- Every codeword $\boldsymbol{y}$ must satisfy the global constraint $\mathcal{A}$.

We define two segments of source data:

- $\boldsymbol{a} = (a_1, \ldots, a_\ell)$, $a_t \in \{0,1\}$.
- $\hat{\boldsymbol{a}} = (\hat{a}_1, \ldots, \hat{a}_{\ell_c})$, $\hat{a}_t \in \{0,1\}$.

The total number of encoded source bits, $\ell + \ell_c$, is referred to as the throughput of the code.

The encoder generates the control word $\boldsymbol{c} = (c_1, \ldots, c_k)$, $c_t \in \{0, \ldots, K-1\}$, by applying a bijective mapping $g : \{0,1\}^{\ell_c} \to C$, where C is a constrained code. The ℓ-bit source data, $\boldsymbol{a}$, are partitioned into k segments $\boldsymbol{a}_t$ of ν_{c_t} bits, $1 \leq t \leq k$. Clearly, we demand $(\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_k) = \boldsymbol{a}$ and thus $\sum_{i=1}^{k} \nu_{c_i} = \ell$. The set of allowed vectors $\boldsymbol{c}$ is

$$C = \{\boldsymbol{c} \in \{0, \ldots, K-1\}^k : (f_{c_1}(\boldsymbol{a}_1), f_{c_2}(\boldsymbol{a}_2), \ldots, f_{c_k}(\boldsymbol{a}_k)) \in \mathcal{S}_{\mathcal{A}} \wedge \sum_{i=1}^{k} \nu_{c_i} = \ell\}. \quad (1)$$

This allows up to $\ell_c = \lfloor \log_2 |C| \rfloor$ additional source bits to be encoded into the control vector $\boldsymbol{c}$ via the mapping $g(\cdot)$. Consequently, the total number of source bits that can be represented in the n-bit codeword $\boldsymbol{y}$ is $\ell + \ell_c$. Each of the k segments of the original source data, denoted $\boldsymbol{a}_t$, is mapped to an m-bit constrained subword $\boldsymbol{y}_t$ using the mapping $\boldsymbol{y}_t = f_{c_t}(\boldsymbol{a}_t)$ for $1 \leq t \leq k$. These subwords are then concatenated to form the complete codeword $\boldsymbol{y}$, which is subsequently transmitted or stored. Importantly, the control vector $\boldsymbol{c}$ is not transmitted to the receiver. However, since the subword sets S_i are pairwise disjoint, the decoder can uniquely identify which mapping was used for each $\boldsymbol{y}_t$. This ensures that both parts of the source data, $\boldsymbol{a}$ and $\hat{\boldsymbol{a}}$, can be fully and uniquely reconstructed from the received codeword $\boldsymbol{y}$.

Figures 1 and 2 depict a block diagram of the encoder and decoder, respectively.

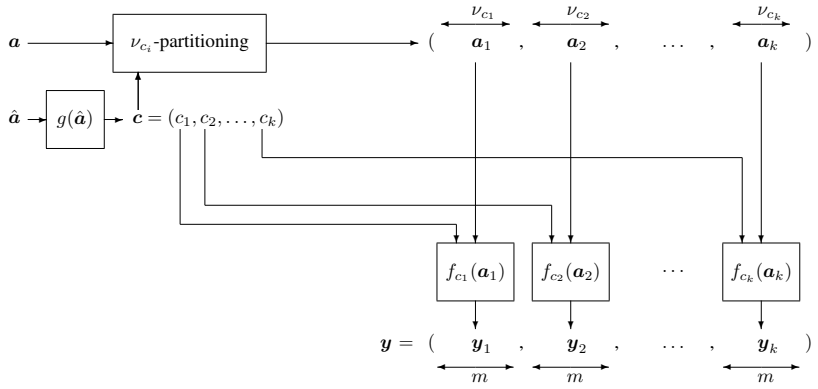

Figure 1. Basic block diagram of a concatenated constrained code. The source data are partitioned into two parts, denoted by $\hat{\boldsymbol{a}}$ and $\boldsymbol{a}$. The portion $\hat{\boldsymbol{a}}$ is mapped to a constrained codeword $\boldsymbol{c}$ by the mapping $\boldsymbol{c} = g(\hat{\boldsymbol{a}})$. The remaining source data $\boldsymbol{a}$ are partitioned, under the control of $\boldsymbol{c}$, into k segments $\boldsymbol{a}_t$ of length ν_{c_t}, $t = 1, 2, \ldots, k$, and are subsequently translated into the final codeword $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)$ using functions $\boldsymbol{y}_t = f_{c_t}(\boldsymbol{a}_t)$ for $t = 1, 2, \ldots, k$. The composite codeword $\boldsymbol{y}$ is sent to the receiver while the auxiliary word $\boldsymbol{c}$ is discarded. Although the block diagram shows k separate look-up tables for conceptual clarity, in practice only $K < k$ distinct look-up tables are required, as they can be reused via multiplexing.

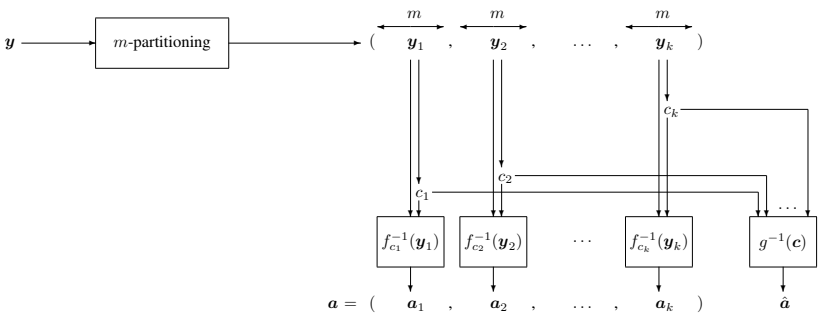

Figure 2. Block diagram of a decoder for a concatenated constrained code. The received vector $\boldsymbol{y}$, of length $n = km$, is first partitioned into k subwords $\boldsymbol{y}_i$, each of length m. Using the appropriate inverse look-up tables, the decoder recovers the control vector $\boldsymbol{c}$ and the data segments $\boldsymbol{a}_t$ via the relation $\boldsymbol{a}_t = {f_{c_t}}^{-1}(\boldsymbol{y}_t)$ for $t = 1, \ldots, k$. Subsequently, the remaining source data $\hat{\boldsymbol{a}}$ are reconstructed by applying the inverse mapping $\hat{\boldsymbol{a}} = g^{-1}(\boldsymbol{c})$.

2.3. Complexity Issues

The encoder begins by mapping a segment of ℓ_c source bits into a control codeword $\boldsymbol{c}$ using the bijective function $\boldsymbol{c} = g(\hat{\boldsymbol{a}})$. It then proceeds to convert each of the k source subwords $\boldsymbol{a}_t$, each of length v_{c_t} for $t = 1, 2, \ldots, k$, into corresponding m-bit constrained subwords $\boldsymbol{y}_t$ via the mapping $\boldsymbol{y}_t = f_{c_t}(\boldsymbol{a}_t)$.

Assuming the k conversions $\boldsymbol{y}_t = f_{c_t}(\boldsymbol{a}_t)$ can be implemented in a multiplexed fashion, the primary hardware requirements for encoding and decoding a concatenated constrained code are limited to the following:

- The K look-up tables for the sets S_i, where $0 \leq i \leq K - 1$.
- A look-up table for the mapping function $g(\cdot)$ that generates $\boldsymbol{c}$.

Unlike traditional approaches that rely on a single large n-bit look-up table, the maximum codeword length n achievable with the proposed concatenated constrained coding scheme is governed by the *product* of the practical maximum sizes of the K look-up tables S_i and the control codebook C. This significantly relaxes the constraints on memory size and look-up complexity.

A compelling example of this technique is the design of a binary constant-weight code, where each $n = 128$-bit codeword contains exactly 32 ones. This design is efficiently realizable using just three modest-sized look-up tables, highlighting the practicality of the concatenated constrained coding approach.

Example 1. *We design an encoder that converts arbitrary source data into binary codewords,* $\boldsymbol{y}$, $y_i \in \{0, 1\}$, *of length n = 128, where each codeword has 32 1's and 96 0's. The maximum number of source bits that can be accommodated is* $\lfloor \log_2 \binom{128}{32} \rfloor = 100$, *where* $\lfloor \cdot \rfloor$ *denotes the truncation or floor function. Therefore, any code constructed for this setting must have a redundancy of at least 28 bits. Since n = 128 is too large to be handled by a single look-up table, we partition the codeword into* $k = 16$ *subwords, each of length m = 8 bits. While other values for k and m are possible, the chosen values simplify manual calculations. In the prior art block code design of multiply constant-weight codes [16] or constant subblock-composition code [17], each subword contains exactly two 1's, and the* $k = 16$ *subwords can be freely cascaded with each other to form a codeword. Clearly, there are* $\binom{8}{2} = 28$ *possible 8-bit subwords. Twelve excess subwords are deleted, so that each subword can convey four source bits, and hence, the 128-bit codewords can carry* $16 \times 4 = 64$ *source bits, resulting in 64 bits of redundancy.*

Using the concatenated constrained code construction, we define $K = 2$ *types of subword sets, namely the set of 8-bit words with a single 1, that is*

$$S_0 = \left\{ \boldsymbol{x} \in \{0,1\}^8 : \sum_{i=1}^{8} x_i = 1 \right\}$$

and the set of 8-bit words with three 1's, namely

$$S_1 = \left\{ \boldsymbol{x} \in \{0,1\}^8 : \sum_{i=1}^{8} x_i = 3 \right\}.$$

We easily find that $|S_0| = \binom{8}{1} = 8$ *and* $|S_1| = \binom{8}{3} = 56$, *where* $|X|$ *denotes the cardinality of the set* X. *We delete* $56 - 32 = 24$ *excess subwords from* S_1 *(keeping 32 subwords). Consequently, we have* $\nu_0 = 3$ *and* $\nu_1 = 5$. *Define the bijective mappings* $f_0 : \{0,1\}^3 \to S_0$ *and* $f_1 : \{0,1\}^5 \to S_1$. *Each 128-bit codeword contains 32 ones by selecting eight subwords from* S_0 *and eight from* S_1. *The* $k = 16$ *subwords together convey* $\ell = 8 \times (3+5) = 64$ *source bits, matching the number of source bits in the prior art; so nothing is gained or lost so far. The k(=16)-bit word,* $\boldsymbol{c}$, *whose elements determine whether to apply* f_0 *or* f_1, *has equal numbers of 1's and 0's. The word,* $\boldsymbol{c}$, *can therefore convey* $\ell_c = \lfloor \log_2 \binom{16}{8} \rfloor = 13$ *source bits. As a result, the total, combined, throughput is* $\ell + \ell_c = 64 + 13 = 77$.

Alternatively, we may choose $m = 16$ *and* $k = 8$, *which requires larger look-up table for* S_0 *and* S_1. *In this case, the traditional approach yields a throughput of* $8 \times 10 = 80$ *bits. In contrast, the concatenated constrained code construction achieves a throughput of* $4(9+12)+6 = 90$ *bits, that is, 90% of the maximum possible.*

The above example shows that the new scheme leverages both subword composition and control-word combinatorics to improve data throughput beyond traditional designs.

In the next sections, we exploit the concatenated constrained code for the construction of efficient constant-weight and low-weight codes. We start with a review of the state of the art.

3. Constant-Weight Codes, Preliminaries, Redundancy, Prior Art

3.1. Introduction

In a binary constant-weight code, each codeword has the same number of 1's. These codes, also known as 'm-out-of-n' [7], codes, have found applications across a wide range of devices and systems including data storage [2], code-division multiple-access (CDMA) systems for optical fibers [18], test pattern generation for circuit testing [19], identification coding [20], and VLSI design [21,22]. Traditionally, constant-weight codes have been employed in data transmission and data storage systems that suffer from low-frequency noise or are subject to low-frequency bandwidth constraints [2,23,24]. More recently, they have been proposed for use in DNA-based storage systems. Knuth's celebrated implementation [25,26] generates 'balanced' codewords i.e., codewords of weight $w = \frac{n}{2}$, n even, with a computational complexity that scales with n, and a redundancy that is approximately twice the theoretical minimum for large n.

A simple and efficient algorithm for producing a codeword with a prescribed imbalance is not available, and the development of such an algorithm remains an open research problem [22,27]. Designing low-complexity encoder/decoder architectures for very high-rate constant-weight codes is of significant practical and economic value. There is a clear demand for efficient constant-weight codes that avoid the need of exorbitantly large look-up tables while maintaining high performance.

3.2. Redundancy

Let $\boldsymbol{x} = (x_1, \ldots, x_n)$, $x_i \in \{0,1\}$, be a binary word of length n. The *Hamming weight* of $\boldsymbol{x}$, denoted by $w(\boldsymbol{x})$, is defined by $w(\boldsymbol{x}) = \sum_{i=1}^{n} x_i$. A constant-weight code, denoted by $\mathcal{S}(w,n)$, defined by

$$\mathcal{S}(w,n) = \{\boldsymbol{x} \in \{0,1\}^n : w(\boldsymbol{x}) = w\}, \tag{2}$$

comprises all binary words of length n and weight w, $0 \leq w \leq n$. The size of $\mathcal{S}(w,n)$, denoted by $|\mathcal{S}(w,n)|$, equals

$$|\mathcal{S}(w,n)| = \binom{n}{\gamma n}, \tag{3}$$

where $\gamma = \frac{w}{n}$, $0 \leq \gamma \leq \frac{1}{2}$, denotes the *relative weight*. In case we have a code with codewords with equal numbers of 1's and 0's, i.e., $2w = n$, n even, the code is said to be *balanced*. Note that without loss of generality we study the case $\gamma \leq 1/2$ as the case $\gamma > 1/2$ is simply found by inverting (flipping) the binary symbols of a codeword.

The *capacity* and *maxentropic* redundancy of constant-weight codes, denoted by $C(w,n)$ and $\rho(w,n)$, are defined by

$$C(w,n) = \log_2 |\mathcal{S}(w,n)| = \log_2 \binom{n}{\gamma n} \tag{4}$$

and

$$\rho(w,n) = n - C(w,n). \tag{5}$$

Using [28], we find the useful approximation

$$\rho(w,n) \approx \frac{1}{2}\log_2 n + (1 - H(\gamma))n + \frac{1}{2}\log_2 \frac{\pi}{2},\ n \gg 1, \tag{6}$$

where Shannon's *entropy* function, $H(x)$, is defined by

$$H(x) = -x\log_2(x) - (1-x)\log_2(1-x). \tag{7}$$

The maxentropic redundancy, $\rho(w,n)$, is applied as a yardstick to evaluate the performance of implemented codes.

3.3. Traditional Code Design Approach

In the prior art, see [29] and the references therein, a codeword, $\boldsymbol{y}$, of length n is divided into k constrained subwords, $\boldsymbol{y}_i$, $1 \leq i \leq k$, of equal length m, where $n = km$. The weight of each subword, $\boldsymbol{y}_i$, is prescribed and denoted by $\omega_i = w(\boldsymbol{y}_i)$, $\omega_i \in \{0, \ldots, m\}$, where $\sum_{i=1}^{k} \omega_i = w$. The weight distribution vector is denoted by $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_k)$. The redundancy, denoted by $\rho_1(w,n)$, equals, see (5),

$$\rho_1(w,n) = \sum_{i=1}^{k} \rho(\omega_i, m). \tag{8}$$

Note that $\rho_1(w,n)$ describes a lower bound to the redundancy of implemented codes as in practice the (sub)code sizes $\binom{m}{\omega_i}$ are truncated to the nearest power of two, see Example 1 for an illustration. We have opted not to truncate the code sizes in our computations, except in the worked design examples, in order to maintain analytical tractability. The *efficiency* of this construction, denoted by η_1, is defined as [29]

$$\eta_1 = \frac{\rho(w,n)}{\rho_1(w,n)}. \tag{9}$$

It is shown in the Appendix A that a uniform, or *flat*, weight distribution minimizes the redundancy.

Numerical results for the efficiency of the prior art construction, η_1, are presented in Table 1. Additional efficiency parameters, η_2 and η_3, for the new constructions are introduced and discussed in Sections 4.2 and 4.3, respectively.

Table 1. Code efficiency, η_1, η_2, and η_3 versus codeword length n, weight w, and number of subwords k, where $m = n/k$, $v = w/k$, and $\gamma = w/n$. The parameter η_1 refers to the efficiency of the prior art code, see (9), while the parameters η_2 and η_3 refer to the code efficiency of the new constructions, see (24) and (33).

n	w	k	m	v	γ	η_1	η_2	η_3
900	240	60	15	4	0.2667	0.5528	0.6560	0.7792
900	240	30	30	8	0.2667	0.6742	0.7533	0.8195
900	240	12	75	20	0.2667	0.8165	0.8593	0.8864
900	240	6	150	40	0.2667	0.8966	0.9192	0.9317
600	240	120	5	2	0.4000	0.1109	0.1539	0.3828
600	240	60	10	4	0.4000	0.1629	0.2309	0.3778
600	240	30	20	8	0.4000	0.2436	0.3248	0.4277
600	240	24	25	10	0.4000	0.2771	0.3601	0.4528
600	240	12	50	20	0.4000	0.4066	0.4879	0.5540
600	240	6	100	40	0.4000	0.5705	0.6380	0.6814
300	60	30	10	2	0.2000	0.6473	0.7334	0.8714
300	60	12	25	5	0.2000	0.7842	0.8435	0.8943
300	60	6	50	10	0.2000	0.8715	0.9058	0.9283

In the next section, we apply the concatenated constrained code to the design of constant-weight codes.

4. Concatenated Constant-Weight Code

4.1. Concatenated Constrained Code

As in Section 3.3, the n-bit codeword $\boldsymbol{y}$ is partitioned into k subwords, $\boldsymbol{y}_i$, $1 \leq i \leq k$, of equal length m. We have $w(\boldsymbol{y}_i) = \omega_i$, $\omega_i \in \{0, \ldots, m\}$ and $\sum_{i=1}^{k} \omega_i = w$. In the prior art, the subword weights, $\omega_i = w(\boldsymbol{y}_i)$, are assumed to be fixed for the whole transmission. In this section, however, the weights of the subwords are controlled, 'modulated' in engineering terms, by a constrained codeword $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_k)$, which is taken from the predefined set Ω. Note that $\boldsymbol{\omega}$ and Ω play the same role as $\boldsymbol{c}$ and C in Section 2.

The coding of $\boldsymbol{y}$ is carried out in two distinct steps. First, source data are mapped bijectively to the weight distribution vector $\boldsymbol{\omega} \in \Omega$ via a look-up table. In the second step, the encoder uses the look-up tables corresponding to the constant-weight codes $\mathcal{S}(\omega_i, m)$, $1 \leq i \leq k$, to convert the source data into the subwords $\boldsymbol{y}_i$. Since the mapping ω_i to $\mathcal{S}(\omega_i, m)$, $1 \leq i \leq k$, is bijective, the decoder can uniquely determine the vector $\boldsymbol{\omega}$, and thereby recover the original source data from the received $\boldsymbol{y}$.

Define the vector $\boldsymbol{u} = (u_0, u_1, \ldots, u_m)$, $u_j \in \{0, \ldots, k\}$, where u_j represents the number of occurrences of the symbol j in $\boldsymbol{\omega}$. The vector $\boldsymbol{u}$ is commonly referred to as the *histogram* of $\boldsymbol{\omega}$. An allowed vector $\boldsymbol{u}$ must satisfy

$$\sum_{i=0}^{m} u_i = k \text{ and } \sum_{i=0}^{m} i u_i = w. \tag{10}$$

The set Ω is the set of k-vectors, $\boldsymbol{\omega}$, of fixed composition $(u_0, u_1, \ldots, u_m)$, that is, the number of 0's, 1's, $\ldots, m$'s in $\boldsymbol{\omega}$ is given by $u_0, u_1, \ldots, u_m$, respectively. The size of the constant composition code Ω is

$$|\Omega| = \frac{k!}{u_0! u_1! \ldots u_m!}. \tag{11}$$

The number of source words that can be accommodated, denoted by M_Ω, equals

$$M_\Omega = \frac{k!}{u_0! u_1! \ldots u_m!} \prod_{i=0}^{m} \binom{m}{i}^{u_i}. \tag{12}$$

Below we illustrate and analyze a design of two simple cases, where $\boldsymbol{u}$ contains $K = 2$ or $K = 3$ non-zero elements, which are denoted here by the binary and ternary case. We start with a description of the binary case, $K = 2$.

4.2. Binary Case, $K = 2$

Let $\omega_i \in \{v_0, v_1\}$, so that $u_{v_0} \neq 0$ and $u_{v_1} \neq 0$, and otherwise $u_i = 0$. We obtain from (10) that

$$u_{v_0} + u_{v_1} = k \text{ and } v_0 u_{v_0} + v_1 u_{v_1} = w. \tag{13}$$

There are manifold solutions to the above system of (positive) integer equations. An enticing option for k even is $v_0 = v - 1$ and $v_1 = v + 1$, so that $u_{v-1} = u_{v+1} = \frac{k}{2}$, and

$$\Omega_2 = \{\boldsymbol{\omega} \in \{v-1, v+1\}^k : \sum_{i=1}^{k} \omega_i = w\} \tag{14}$$

denotes the set of allowed vectors $\boldsymbol{\omega}$.

The relationship between $\boldsymbol{\omega}$ and $\boldsymbol{c}$ is a straightforward renumbering of their elements by

$$c_i = (\omega_i - v + 1)/2, \;\; i = 1, \ldots, k, \tag{15}$$

so that $c_i \in \{0, 1\}$, and renaming $S_0 = \mathcal{S}(v-1, m)$ and $S_1 = \mathcal{S}(v+1, m)$. The coding circuitry comprises two look-up tables, S_0 and S_1, of width m, plus a (binary) look-up table for the balanced code, C, of width k. Note that Knuth's algorithm can be used for mounting k, when the look-up table for $\boldsymbol{c}$ is uneconomically expensive.

We obtain from (14) that

$$|\Omega_2| = \binom{k}{\frac{k}{2}}, \tag{16}$$

so that, see (12),

$$M_{\Omega_2} = |\mathcal{S}(v-1, m)|^{\frac{k}{2}} |\mathcal{S}(v+1, m)|^{\frac{k}{2}} \binom{k}{\frac{k}{2}}. \tag{17}$$

Hence the redundancy, denoted by $\rho_2(w, n)$, is

$$\rho_2(w, n) = n - \frac{k}{2} \log_2 \binom{m}{v-1} \binom{m}{v+1} - \log_2 \binom{k}{\frac{k}{2}}.$$

Since

$$\binom{m}{v-1} = \frac{v}{m-v+1} \binom{m}{v} \tag{18}$$

and

$$\binom{m}{v+1} = \frac{m-v}{v+1} \binom{m}{v}, \tag{19}$$

we obtain

$$\binom{m}{v-1}\binom{m}{v+1} = \frac{v(m-v)}{(v+1)(m-v+1)}\binom{m}{v}^2, \tag{20}$$

so that

$$\begin{aligned}\rho_2(w,n) &= k\left(\rho(v,m)+\beta(v,m)-\frac{1}{k}\log_2\binom{k}{\frac{k}{2}}\right)\\ &= \rho_1(v,m)+k\beta(v,m)-\log_2\binom{k}{\frac{k}{2}},\end{aligned} \tag{21}$$

where

$$\beta(v,m) = -\frac{1}{2}\log_2\frac{v(m-v)}{(v+1)(m-v+1)}. \tag{22}$$

Figure 3 shows the coefficient $\beta(v,m)$ versus v, $1 \le v \le m/2$, for $m = 16, 24$, and 32.

The coefficient $\beta(v,m)$ is an important parameter as it quantifies the additional redundancy per subword required for the concatenated coding system. Note in (21) that, with respect to the redundancy of the traditional code, $\rho_1(w,n)$, we have two additional terms in $\rho_2(w,n)$. On one hand we add redundancy, namely $\beta(v,m)$ bit per subword, by deviating from the flat weight distribution, as discussed in Appendix A. On the other hand, we reduce redundancy, namely

$$\frac{1}{k}\log_2\binom{k}{\frac{k}{2}} \approx 1-\frac{1}{2k}\log_2(k) \tag{23}$$

bits per subword by encoding data in the second (binary) code C. Figure 3 shows that the loss, $\beta(v,m)$, is around 0.1-0.2 bits per subword over a wide range of the parameters v and m.

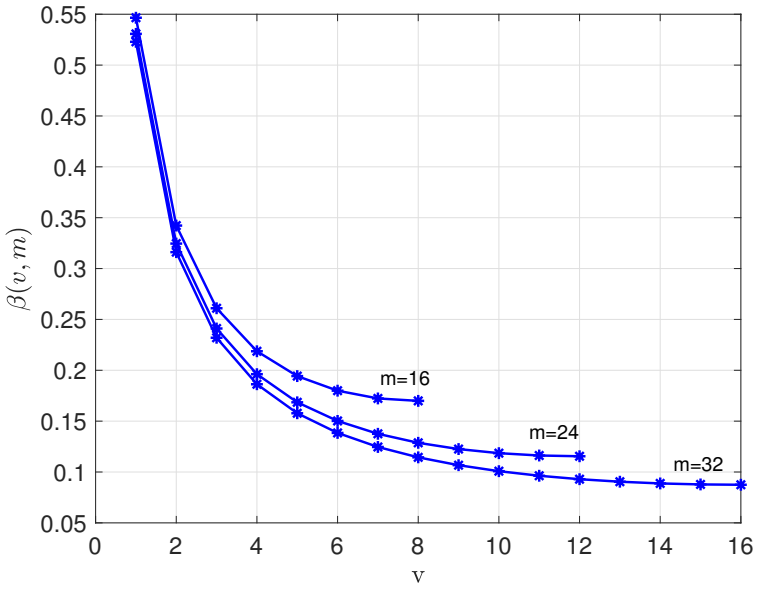

Figure 3. Coefficient $\beta(v,m)$ versus v for $m = 16, 24$, and 32.

The efficiency of codes based on the binary concatenated constrained code is defined by

$$\eta_2 = \frac{\rho(w,n)}{\rho_2(w,n)}. \tag{24}$$

Table 1 presents numerical results for the code's efficiency, η_2. We observe a notable improvement in efficiency compared to that of the traditional scheme, η_1, especially for larger values of k.

4.3. Ternary Case, $K = 3$

For the ternary case, we define the values of u_i by

$$u_{v-1} = u_{v+1} = \frac{k-a}{2} \text{ and } u_v = a,\ a = 2, 4, \ldots, k-2 \tag{25}$$

and otherwise $u_i = 0$. Note that for $a = 0$ we have the binary case, $K = 2$, as detailed in the previous subsection. Let Ω_3 denote the constant composition code based on $\boldsymbol{u}$. Then

$$|\Omega_3| = \frac{k!}{\frac{k-a}{2}!a!\frac{k-a}{2}!}, \tag{26}$$

and the number of source words that can be accommodated, denoted by M_{Ω_3}, equals

$$M_{\Omega_3} = |\mathcal{S}(v-1,m)|^{\frac{k-a}{2}}|\mathcal{S}(v,m)|^a|\mathcal{S}(v+1,m)|^{\frac{k-a}{2}}|\Omega_3|. \tag{27}$$

The overall redundancy, denoted by $\rho_3(w,n)$, is

$$\rho_3(w,n) = k\rho(v,m) + (k-a)\beta(v,m)) - \log_2|\Omega_3|. \tag{28}$$

For the special case $a = \frac{k}{3}, k \mod 3 = 0$, we have

$$|\Omega_3| = \frac{k!}{(\frac{k}{3}!)^3} \tag{29}$$

and

$$\rho_3(w,n) = k(\rho(v,m) + \frac{2}{3}\beta(v,m)) - \log_2\frac{k!}{(\frac{k}{3}!)^3}. \tag{30}$$

For $k \gg 1$, we obtain the Stirling approximation [30]

$$\frac{k!}{(\frac{k}{3}!)^3} \approx \frac{3^{k+\frac{3}{2}}}{2\pi k},\ k \gg 1, \tag{31}$$

so that

$$\rho_3(w,n) \approx k(\rho(v,m) + \frac{2}{3}\beta(v,m) - \log_2(3)). \tag{32}$$

Define the efficiency of this construction by

$$\eta_3 = \frac{\rho(w,n)}{\rho_3(w,n)}. \tag{33}$$

Table 1 shows numerical results for the code's efficiency, η_3. As in the binary case, we find $\boldsymbol{c}$ by renumbering the elements of $\boldsymbol{\omega}$ according to $c_i = \omega_i - v + 1$, $i = 1, \ldots, k$, so that $c_i \in \{0,1,2\}$. We then rename the sets as $S_0 = \mathcal{S}(v-1,m)$, $S_1 = \mathcal{S}(v,m)$ and $S_2 = \mathcal{S}(v+1,m)$. It is assumed that a look-up table is used to map the source data $\hat{\boldsymbol{a}}$ into $\boldsymbol{c}$. For larger values of k, where using a look-up table becomes impractical, we may resort to enumeration algorithms for multiset permutations [31] or Knuth-like algorithms [30,32], which can be employed instead. The next example elaborates Example 1 for $K = 3$.

Example 2. *We choose, as in Example 1, $n = 128$, $w = 32$, $v = 2$, and $m = 8$. For the ternary case, $K = 3$, we obtain the following results from Example 1: $\lfloor \log_2(|\mathcal{S}(1,8)|) \rfloor = 3$,*

$\lfloor \log_2(|\mathcal{S}(2,8)|) \rfloor = 4$, *and* $\lfloor \log_2(|\mathcal{S}(3,8)|) \rfloor = 5$. *From this, we can easily determine that the maximum throughput* $\ell + \ell_c = 84$ *is attained for* $\boldsymbol{u} = (0,6,4,6,0,0,0,0,0,0)$ *or* $\boldsymbol{u} = (0,5,6,5,0,0,0,0,0,0)$.

4.4. Error Propagation Effects

Single bit errors in the received codeword can lead to an avalanche of errors during decoding. This phenomenon, called *error propagation*, is especially pronounced in long codes based on enumerative methods [14]. In this subsection, we examine the error propagation behavior of constant-weight codes constructed using concatenated constrained codes.

Let the received codeword be denoted by $\hat{\boldsymbol{y}} = (\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_k)$, which differs from the transmitted codeword, $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)$, in a single (unknown) position. Since $\boldsymbol{y}$ belongs to a constant-weight code, such an error is detectable.

For a constant-weight code constructed with parameter $K = 2$ (binary case), as illustrated in Example 1, each subword $\boldsymbol{y}_i$ belongs to the set $\mathcal{S}(v-1,m) \cup \mathcal{S}(v+1,m)$, $1 \le i \le k$, and $v = w/k$. Since the minimum Hamming distance between elements of this set is two, it is possible to identify the specific subword $\boldsymbol{y}_i$ that was received in error. Assume that the subword $\hat{\boldsymbol{y}}_j$ with index j is in error, i.e., $\hat{\boldsymbol{y}}_j \neq \boldsymbol{y}_j$. By definition, we have $\sum_{i=1}^{k} c_i = k/2$, so the control vector $\boldsymbol{c}$ can be fully decoded by, see (15),

$$c_i = \begin{cases} \frac{(w(\hat{\boldsymbol{y}}_i)-v+1)}{2}, & 1 \le i \le k, i \neq j \\ \frac{k}{2} - \sum_{i=1, i\neq j}^{k} c_i, & i = j. \end{cases} \tag{34}$$

Thus, in the binary case, $K = 2$, all data can be recovered except for the unknown m-bit subword $\hat{\boldsymbol{y}}_j$, which must be marked as an erasure. In the ternary case ($K = 3$), each $\boldsymbol{y}_i$ belongs to the set $\mathcal{S}(v-1,m) \cup \mathcal{S}(v,m) \cup \mathcal{S}(v+1,m)$, $1 \le i \le k$. Since the minimum Hamming distance between its elements is one, it is not always possible to identify the erroneous subword. As a result, the entire n-bit codeword must be flagged as an erasure. Note that for the ternary case we may choose the sets $\mathcal{S}(v-2,m)$, $\mathcal{S}(v,m)$ and $\mathcal{S}(v+2,m)$, so that it is also possible to identify the specific subword $\boldsymbol{y}_i$ that was received in error.

5. Very Long Codes

In the previous sections, we demonstrate that concatenated constrained coding can efficiently produce longer constant-weight codewords compared to conventional state-of-the-art methods. However, the scalability of such codes is limited: the codeword length $n = km$ is, in practice, constrained by the maximum feasible size of the look-up tables, either the subword tables $\mathcal{S}_i$ of width m, or the control code table C of width k.

To overcome these practical limitations, several strategies can be adopted:

- Use enumerative coding techniques to generate the subword sets $\mathcal{S}_i$ and/or the control codebook C.
- Apply a second layer of concatenated constrained coding to generate entries within $\mathcal{S}_i$ and/or C, effectively building a hierarchy of constrained encodings.

The next example illustrates the effectiveness of this method through numerical results for a constant-weight code of length $n = 2048$.

Example 3. *Consider the case where* $n = 2048$ *and* $w = 512$ *(*$\gamma = 1/4$*). The theoretical minimum redundancy for this constant-weight code is* $\rho(512, 2048) = 392.12$ *bits. Due to the large value of n, enumerative coding becomes impractical for this low-weight scenario, so we explore alternative coding strategies:*

- *We can cascade sixteen codewords of length* $n' = 128$ *and* $w' = 32$*, see Example 2, with a total throughput of* $16 \times 84 = 1344$ *bit.*

- *We can, as illustrated in Example 1, select a concatenated constrained code with* $m = 8$ *and* $k = 256$*. Since* $k = 256$ *is too large for directly using a look-up table to generate the control word* $\boldsymbol{c}$*, we alternatively consider applying enumerative coding for generating the balanced codeword* $\boldsymbol{c}$*. Then, the redundancy is five bit, so that the scheme's throughput is* $256 - 5 + k \times 4 = 1275$ *bit. We may apply Knuth's algorithm to generate codeword* $\boldsymbol{c}$ *of length* $k = 256$ *with an 8-bit redundancy, so that the throughput is* $256 - 8 + k \times 4 = 1272$ *bit.*
- *We may construct a long code by using multiple smaller concatenated constrained codes as building blocks. For example, an* $(n = 2048, w = 512)$ *code is constructed by first constructing two concatenated constrained codes with parameters* $n' = 128$ *of weights* $w' = 31$ *and* $w'' = 33$*, respectively, by using* $\boldsymbol{u}' = (0, 6, 5, 5, 0, 0, 0, 0, 0, 0)$ *and* $\boldsymbol{u}'' = (0, 5, 5, 6, 0, 0, 0, 0, 0, 0)$ *instead of* $\boldsymbol{u} = (0, 6, 4, 6, 0, 0, 0, 0, 0, 0)$ *of the* $n' = 128$*,* $w' = 32$ *code detailed in Example 2. Both codes have a throughput of 84 bit. On top of these codes, we define a concatenated constrained code with* $k = 16$ *and* $n' = 128$*, the combined throughput equals* $16 \times 84 + 20 = 1364$ *bit. The code's implementation requires six small look-up tables.*

While numerous other design options exist beyond those discussed above, the examples provided offer a representative insight into the key trade-offs involved.

6. Low-Weight Codes

Low-weight, or sometimes referred to as light-weight or bounded-weight, codes have found applications in efficiently synthesizing deoxyribonucleic acid (DNA) for massive data storage, where multiple DNA strands are synthesized in parallel [33]. Applications can also be found in memristor crossbar arrays for reducing the number of sneak-paths [34], and simultaneous energy and data transfer [17,35].

A low-weight code of length n and weight at most t, $0 \leq t \leq n$, denoted by $\hat{\mathcal{S}}(t, n)$, is defined by the union of the sets of words of weight $w \leq t$,

$$\hat{\mathcal{S}}(t, n) = \bigcup_{w=0}^{t} \mathcal{S}(w, n). \tag{35}$$

The redundancy, denoted by $\hat{\rho}(t, n)$, equals

$$\hat{\rho}(t, n) = n - \log_2 |\hat{\mathcal{S}}(t, n)| = n - \log_2 \sum_{w=0}^{t} \binom{n}{w}. \tag{36}$$

For asymptotically large n we obtain [28]

$$\frac{1}{n}\hat{\rho}(t, n) \approx 1 - H\left(\frac{t}{n}\right), \ \frac{t}{n} < \frac{1}{2}. \tag{37}$$

Figure 4 shows the normalized redundancy, $\hat{\rho}(t, n)/n$, versus relative maximum weight, t/n for $n = 12, 24$, and ∞.

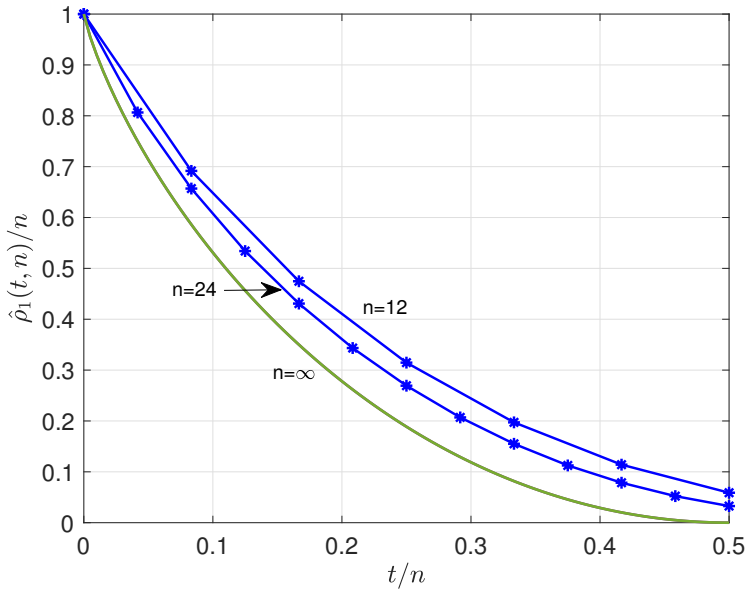

Figure 4. Lower bound normalized redundancy, $\hat{\rho}(t,n)/n$, of low-weight codes versus t/n for $n = 12, 24$, and ∞.

6.1. Code Design

The conventional construction of a low-weight block code follows a similar approach to that of constant-weight codes. The source data and codewords are partitioned into k manageable subwords, with look-up tables used to map the source data into these subwords. As assumed above let $n = km$ be the length of the low-weight codeword, where m is the length of each subword, and $t = kv$. The redundancy of a conventional code, denoted by $\hat{\rho}_1(t,n)$, is

$$\hat{\rho}_1(t,n) = k\hat{\rho}(v,m). \tag{38}$$

In the next subsection, we describe a simple low-weight code based on the concatenated constrained code format.

Binary Case, $K = 2$

The source data are divided into two segments, namely $\hat{\boldsymbol{a}}$ and $\boldsymbol{a}$. The first segment, $\hat{\boldsymbol{a}}$, is translated into a binary constrained codeword, $\boldsymbol{c} = (c_1, \ldots, c_k) = g(\hat{\boldsymbol{a}})$, where $c_i \in \{0,1\}$, of length k. The second segment, $\boldsymbol{a}$, is translated into a series of k weight-constrained m-bit words, each taken from either the set $S_0(v,m)$ or $S_1(v,m)$. For a concatenated constrained coding construction, the sets $S_0(v,m)$ and $S_1(v,m)$ must be disjoint. A convenient choice for these sets is

$$S_0(v,m) = \mathcal{S}(v+1,m) \cup \mathcal{S}(v,m) \tag{39}$$

and

$$S_1(v,m) = \bigcup_{w=0}^{v-1} \mathcal{S}(w,m). \tag{40}$$

Let $\boldsymbol{c} = (c_1, \ldots, c_k)$ be a balanced word, k even, then the n-bit codeword found by cascading k m-bit subwords from $S_{c_i}(v,m)$, $1 \le i \le k$, has a weight less than or equal to t. The redundancy of the code, denoted by $\hat{\rho}_2(t,n)$, equals

$$\hat{\rho}_2(t,n) = n - \frac{k}{2}\log_2 |S_0(v,m)||S_1(v,m)| - \log_2 \binom{k}{\frac{k}{2}}. \tag{41}$$

Define

$$\hat{\beta}(v,m) = -\frac{1}{2}\log_2 \frac{|S_0(v,m)||S_1(v,m)|}{|\hat{\mathcal{S}}(v,m)|^2}, \tag{42}$$

then

$$\hat{\rho}_2(t,n) = k(\hat{\rho}(v,m) + \hat{\beta}(v,m)) - \log_2 \binom{k}{\frac{k}{2}}. \tag{43}$$

Figure 5 shows the coefficient $\hat{\beta}(v,m)$ versus v, $1 \le v \le m/2$, for $m = 16, 24$, and 32.

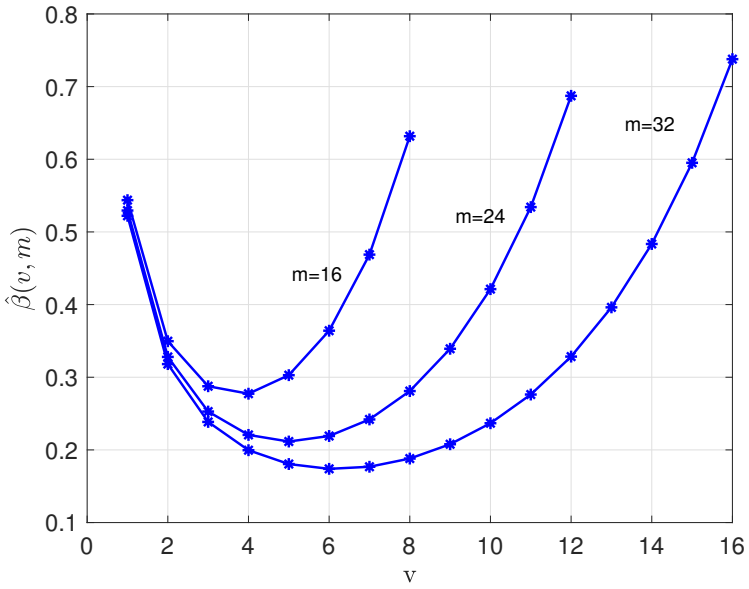

Figure 5. Coefficient $\hat{\beta}(v,m)$ versus v for $m = 16, 24$, and 32.

Define the efficiency parameters as $\hat{\eta}_1 = \hat{\rho}(t,n)/\hat{\rho}_1(t,n)$ and $\hat{\eta}_2 = \hat{\rho}(t,n)/\hat{\rho}_2(t,n)$. Numerical results are presented in Table 2. It can be observed that the concatenated codes reduce the redundancy compared to the baseline by 6.5 percentage points for $k = 128$.

Table 2. Code efficiency, $\hat{\eta}_1$ and $\hat{\eta}_2$ versus number of subwords per codeword, k, for codeword length $n = 1024$, and maximum weight $t = 128$.

k	$m = n/k$	$v = w/k$	$\hat{\eta}_1$	$\hat{\eta}_2$
128	8	1	0.7633	0.8294
64	16	2	0.8283	0.8880
32	32	4	0.8851	0.9246
16	64	8	0.9291	0.9513
8	128	16	0.9600	0.9711

7. Conclusions

We have introduced a new class of constrained codes, referred to as concatenated constrained codes, which enable the construction of very long constrained codewords with significantly reduced complexity. Similar to traditional methods, the source data are divided into smaller blocks. In the first step, one segment of the source data is encoded using a set of small look-up tables, each corresponding to a disjoint set of valid output sequences. In the second step, another segment of the source data is encoded into a control codeword that determines which look-up table is used for each portion of the first segment.

This layered encoding structure enables the generation of longer codewords with lower redundancy compared to conventional approaches, while eliminating the need for massive look-up tables.

We demonstrated the effectiveness of this approach through two case studies focused on constructing binary constant-weight and light-weight codewords of length n, each containing exactly w ones, or not more than w ones, where $w \leq n/2$. The concatenated constrained codes in these examples achieve lower redundancy than leading state-of-the-art solutions, while requiring only three or four compact look-up tables, highlighting both their efficiency and practicality.

We have shown that extremely long codewords can be constructed by applying a second layer of concatenated constrained coding on top of an initial concatenated constrained scheme, effectively generating elements within the sets $\mathcal{S}_i$ and/or the codebook C.

Author Contributions: Conceptualization, Kees Schouhamer Immink, J.H.W., T.T.N. and K.C.; Writing–original draft, K.S.I., J.H.W., T.T.N. and K.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Singapore Ministry of Education Academic Research Fund Tier 2 T2EP50221-0036 and SUTD Kickstarter Initiative (SKI) Grant 2021_04_05.

Data Availability Statement: No new data were created or analyzed in this study.

Conflicts of Interest: Author Kees Schouhamer Immink was employed by the company Turing Machines Inc. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Appendix A

We investigate minimizing the redundancy of the basic design approach, see Section 3.3.

Theorem A1. *The number of words*

$$M_1 = \prod_{i=1}^{k} \binom{m}{\omega_i}$$

is maximized by choosing a flat distribution for $\boldsymbol{\omega}$*, where the subword weights,* ω_i*, are as close to* w/k *as possible. Specifically, this means that the difference between any pair of weights is at most 1.*

Proof. Let i_1 and i_2, $i_1 \neq i_2$, be arbitrarily chosen indices such that $1 \leq i_1, i_2 \leq n$. Then the number of n-sequences is

$$M_1 = \binom{m}{\omega_{i_1}} \binom{m}{\omega_{i_2}} \prod_{i=1, i \neq i_1, i_2}^{k} \binom{m}{\omega_i}. \tag{A1}$$

Suppose $\omega_{i_1} > \omega_{i_2} + 1$. To increase M_1 replace ω_{i_1} by $\omega_{i_1} - 1$ and ω_{i_2} by $\omega_{i_2} + 1$ (noting that the sum $\sum_{i=1}^{k} \omega_i = w$ remains constant). Then

$$\begin{aligned} M_1 &= \binom{m}{\omega_{i_1} - 1} \binom{m}{\omega_{i_2} + 1} \prod_{i=1, i \neq i_1, i_2}^{k} \binom{m}{\omega_i}, \\ &= \frac{\omega_{i_1}}{m - \omega_{i_1} + 1} \frac{m - \omega_{i_2}}{\omega_{i_2} + 1} \prod_{i=1}^{k} \binom{m}{\omega_i}. \end{aligned} \tag{A2}$$

Since $\omega_{i_1} > \omega_{i_2} + 1$, it follows that

$$\frac{\omega_{i_1}}{m - \omega_{i_1} + 1} \frac{m - \omega_{i_2}}{\omega_{i_2} + 1} > 1. \tag{A3}$$

Because i_1 and i_2 were arbitrary, this argument applies to any pair of indices. Therefore, the number of words M_1 is maximized when the weights ω_i are balanced as evenly as possible, i.e., when the difference between any two ω_i is at most 1. □

References

1. Shannon, C.E. A Mathematical Theory of Communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423. [CrossRef]
2. Immink, K.A.S. Innovation in Constrained Codes. *IEEE Commun. Mag.* **2022**, *60*, 20–24. [CrossRef]
3. Marcus, B.H.; Siegel, P.H.; Wolf, J.K. Finite-state Modulation Codes for Data Storage. *IEEE J. Sel. Areas Commun.* **1992**, *10*, 5–37. [CrossRef]
4. Ryabko, B. A General Method for the Development of Constrained Codes. *IEEE Trans. Inf. Theory* **2025**, *71*, 3510–3515. [CrossRef]
5. Modha, D.S.; Marcus, B.H. Art of Constructing Low-complexity Encoders/decoders for Constrained Block Codes. *IEEE J. Sel. Areas Commun.* **2001**, *19*, 589–601. [CrossRef]
6. Stockmeyer, L.; Modha, D.S. Links Between Complexity Theory and Constrained Block Coding. *IEEE Trans. Inf. Theory* **2002**, *48*, 59–88. [CrossRef]
7. Ramabadran, T.V. A Coding Scheme for m-out-of-n Codes. *IEEE Trans. Commun.* **1990**, *38*, 1156–1163. [CrossRef]
8. Schulte, P.; Böcherer, G. Constant Composition Distribution Matching. *IEEE Trans. Inf. Theory* **2016**, *62*, 430–434. [CrossRef]
9. Cover, T.M. Enumerative Source Coding. *IEEE Trans. Inf. Theory* **1973**, *19*, 73–77. [CrossRef]
10. Schalkwijk, J.P.M. An Algorithm for Source Coding. *IEEE Trans. Inf. Theory* **1972**, *18*, 395–399. [CrossRef]
11. Hareedy, A.; Calderbank, R. LOCO Codes: Lexicographically-Ordered Constrained Codes. *IEEE Trans. Inf. Theory* **2020**, *66*, 3572–3589. [CrossRef]
12. Kurmaev, O. Constant-Weight and Constant-Charge Binary Run-Length Limited Codes. *IEEE Trans. Inf. Theory* **2011**, *57*, 4497–4515. [CrossRef]
13. Butler, J.T.; Sasao, T. Fast constant weight codeword to index converter. In Proceedings of the 2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS), Seoul, Republic of Korea, 7–10 August 2011; pp. 1–4. [CrossRef]
14. Immink, K.A.S.; Janssen, A.J.E.M. Error propagation assessment of enumerative coding schemes. *IEEE Trans. Inf. Theory* **1999**, *45*, 2591–2594. [CrossRef]
15. Forney, G.D., Jr. *Concatenated Codes*; MIT Press: Cambridge, MA, USA, 1966.
16. Chee, Y.M.; Kiah, H.M.; Zhang, H.; Zhang, X. Constructions of Optimal and Near-Optimal Multiply Constant-Weight Codes. *IEEE Trans. Inf. Theory* **2017**, *63*, 3621–3629. [CrossRef]
17. Tandon, A.; Motani, M.; Varshney, L.R. Subblock-Constrained Codes for Real-Time Simultaneous Energy and Information Transfer. *IEEE Trans. Inf. Theory* **2016**, *62*, 4212–4227. [CrossRef]
18. Chung, F.R.K.; Salehi, J.A.; Wei, V.K. Optical orthogonal codes: Design, analysis and applications. *IEEE Trans. Inf. Theory* **1989**, *35*, 595–604. [CrossRef]
19. Tang, D.T.; Woo, L.S. Exhaustive Test Pattern Generation with Constant Weight Vectors. *IEEE Trans. Comput.* **1983**, *32*, 1145–1150. [CrossRef]
20. Gunlu, O.; Kliewer, J.; Schaefer, R.F.; Sidorenko, V. Code Constructions and Bounds for Identification via Channels. *IEEE Trans. Commun.* **2022**, *70*, 1486–1496. [CrossRef]
21. Tallini, L.G.; Bose, B. Design of balanced and constant weight codes for VLSI systems. *IEEE Trans. Comput.* **1998**, *47*, 556–572. [CrossRef]
22. Skachek, V.; Immink, K.A.S. Constant Weight Codes: An Approach Based on Knuth's Balancing Method. *IEEE J. Sel. Areas Commun.* **2014**, *32*, 908–918. [CrossRef]
23. Cattermole, K.W. *Principles of Pulse Code Modulation*; Iliffe Books Ltd.: London, UK, 1969.
24. Marcovich, S.; Etzion, T.; Yaakobi, E. On Hierarchies of Balanced Sequences. *IEEE Trans. Inf. Theory* **2023**, *69*, 2923–2939. [CrossRef]
25. Knuth, D.E. Efficient Balanced Codes. *IEEE Trans. Inf. Theory* **1986**, *32*, 51–53. [CrossRef]
26. Tallini, L.G.; Bose, B. Balanced codes with parallel encoding and decoding. *IEEE Trans. Comput.* **1999**, *48*, 794–814. [CrossRef]
27. Dao, D.T.; Kiah, H.M.; Nguyen, T.T. Efficient Encoding of Binary Constant-Weight Codes: Variable-Length Balancing Schemes a la Knuth. *IEEE Trans. Inf. Theory* **2024**, *70*, 4731–4746. [CrossRef]
28. Stanica, P. Good Lower and Upper Bounds on Binomial Coefficients. *J. Inequalities Pure Appl. Math.* **2001**, *2*, 30.
29. Tian, C.; Vaishampayan, V.A.; Sloane, N. A Coding Algorithm for Constant Weight Vectors: A Geometric Approach Based on Dissections. *IEEE Trans. Inf. Theory* **2009**, 55, 1051–1060. [CrossRef]
30. Weber, J.H.; Immink, K.A.S.; Siegel, P.H.; Swart, T.G. Perspectives on Balanced Sequences. *arXiv* **2013**, arXiv:1301.6484. [CrossRef]
31. Milenkovic, O.; Vasic, B. Permutation (d,k) codes: Efficient enumerative coding and phrase length distribution shaping. *IEEE Trans. Inf. Theory* **2000**, *46*, 2671–2675. [CrossRef]

32. Mascella, R.; Tallini, L.G. Efficient m-ary Balanced Codes which Are Invariant under Symbol Permutation. *IEEE Trans. Comput.* **2006**, *55*, 929–946. [CrossRef]
33. Immink, K.A.S.; Cai, K.; Nguyen, T.T.; Weber, J.H. Constructions and Properties of Efficient DNA Synthesis Codes. *IEEE Trans. Mol. Biol. Multi-Scale Commun.* **2024**, *10*, 289–296. [CrossRef]
34. Cassuto, Y.; Kvatinsky, S.; Yaakobi, E. Information-Theoretic Sneak-Path Mitigation in Memristor Crossbar Arrays. *IEEE Trans. Inf. Theory* **2016**, *62*, 4801–4813. [CrossRef]
35. Tandon, A.; Kiah, H.M.; Motani, M. Bounds on the Size and Asymptotic Rate of Subblock-Constrained Codes. *IEEE Trans. Inf. Theory* **2018**, *64*, 6604–6619. [CrossRef]