A Coding Algorithm for Constant Weight Vectors: A Geometric Approach Based on Dissections

Chao Tian, Member, IEEE, Vinay A. Vaishampayan Senior Member, IEEE and N. J. A. Sloane Fellow, IEEE

Abstract—We present a novel technique for encoding and decoding constant weight binary vectors that uses a geometric interpretation of the codebook. Our technique is based on embedding the codebook in a Euclidean space of dimension equal to the weight of the code. The encoder and decoder mappings are then interpreted as a bijection between a certain hyper-rectangle and a polytope in this Euclidean space. An inductive dissection algorithm is developed for constructing such a bijection. We prove that the algorithm is correct and then analyze its complexity. The complexity depends on the weight of the vector, rather than on the block length as in other algorithms. This approach is advantageous when the weight is smaller than the square root of the block length.

Index Terms—Constant weight codes, encoding algorithms, dissections, polyhedral dissections, bijections, mappings, Dehn invariant.

I. INTRODUCTION

We consider the problem of encoding and decoding binary vectors of constant Hamming weight \( w \) and block length \( n \). Such sets of vectors (hereafter referred to as codes) are useful in a variety of applications: a few examples are fault-tolerant circuit design and computing [15], pattern generation for circuit testing [25], identification coding [28] and optical overlay networks [26].

The problem of interest is that of designing the encoder and decoder, i.e., the problem of mapping all binary (information) vectors of a given length onto a subset of length-\( n \) vectors of constant Hamming weight \( w \) in a one-to-one manner. In this work, we propose a novel geometric method in which information and code vectors are represented by vectors in \( w \)-dimensional Euclidean space, covering polytopes for the two sets are identified, and a one-to-one mapping is established by dissecting the covering polytopes in a specific manner. This approach results in an invertible integer-to-integer mapping, thereby ensuring unique decodability. The proposed algorithm has a natural recursive structure, and an inductive proof is given for unique decodability. The issue of efficient encoding and decoding is also addressed. We show that the proposed algorithm has complexity \( O(w^2) \), where \( w \) is the weight of the codeword, independent of the codeword length \( n \).

Compared to a linear-time encoding algorithm [16], the algorithm presented here is faster only when \( w = O(\sqrt{n}) \), in which case the rate of the resulting code, \( (1/n) \log \binom{n}{w} \), approaches zero as \( n \to \infty \). However, we view this paper as the starting point of a new approach to the classic problem of encoding and decoding constant weight codes, and we believe there is considerable room for improvement. Some steps in this direction can be found in [22], [27]. Further, the geometric view advanced here is also of potential significance to areas such as source coding and simulation.

Dissections are of considerable interest in geometry, partly as a source of puzzles, but more importantly because they are intrinsic to the notion of volume. Of the 23 problems posed by David Hilbert at the International Congress of Mathematicians in 1900, the third problem dealt with dissections. Hilbert asked for a proof that there are two tetrahedra of the same volume with the property that it is impossible to dissect one into a finite number of pieces that can be rearranged to give the other, i.e., that the two tetrahedra are not equidecomposable.

The problem was immediately solved by Dehn [6]. In 1965, after 20 years of effort, Sydler [24] completed Dehn’s work. The Dehn-Synyder theorem states that a necessary and sufficient condition for two polyhedra to be equidecomposable is that they have the same volume and the same Dehn invariant. This invariant is a certain function of the edge-lengths and dihedral angles of the polyhedron. An analogous theorem holds in four dimensions (Jessen [11]), but in higher dimensions it is known only that equality of the Dehn invariants is a necessary condition. In two dimensions any two polygons of equal area are equidecomposable, a result due to Bolyai and Gerwein (see Boltianskii [1]). Among other books dealing with the classical dissection problem in two and three dimensions we mention in particular Frederickson [7], Lindgren [13] and Sah [19].

The remainder of the paper is organized as follows. We provide background and review relevant previous work in Section II. Section III describes our geometric approach and gives algorithms then given in Section IV, and the correctness of the algorithms is established. Section V summarizes the paper.

II. BACKGROUND AND PREVIOUS METHODS

Let us denote the Hamming weight of a length-\( n \) binary sequence \( s := (s_1, s_2, \ldots, s_n) \) by \( w(s) := |\{i : s_i = 1\}| \), where \(|\cdot|\) is the cardinality of a set.

**Definition 1:** An \((n, w)\) constant weight binary code \( C \) is a set of length-\( n \) sequences such that any sequence \( s \in C \) has weight \( w(s) = w \).

If \( C \) is an \((n, w)\) constant weight code, then its rate \( R := (1/n) \log_2 |C| \leq R(n, w) := (1/n) \log_2 \binom{n}{w} \). For fixed \( \beta \) and \( w = \lfloor \beta n \rfloor \), we have

\[
\overline{R} := \lim_{n \to \infty} R(n, w) = h(\beta),
\]

where \( h(\beta) := -\beta \log_2(\beta) - (1 - \beta) \log_2(1 - \beta) \) is the entropy function. Thus \( \overline{R} \) is maximized when \( \beta = 1/2 \), i.e., the asymptotic rate is highest when the code is balanced.

The authors are with AT&T Shannon Laboratory, Florham Park, NJ.
The (asymptotic) efficiency of a code relative to an infinite-length code with the same weight to length ratio \( w/n \), given by 
\[
\eta := R/R, \text{ can be written as } \eta = \eta_1 \eta_2 \text{ where } \eta_1 := R/R(n, w) \text{ and } \eta_2 := R(n, w)/R.
\]
The first term, \( \eta_1 \), is the efficiency of a particular code relative to the best possible code with the same length and weight; the second term, \( \eta_2 \), is the efficiency of the best finite-length code relative to the best infinite-length code.

From Stirling’s formula we have
\[
\eta \approx 1 - \frac{\log_2(2\pi n \beta(1-\beta))}{2nh(\beta)}.
\]
A plot of \( \eta \) as a function of \( n \) is given in Fig. 1 for \( \beta = 1/2 \). The slow convergence visible here is the reason one needs codes with large block lengths.

The problem of finding efficient algorithms for encoding and decoding constant weight vectors has been considered by several authors. We briefly discuss two previous methods that are relevant to our work. The first, a general purpose technique based on the idea of lexicographic ordering and enumeration of codewords in a codebook (Schalkwijk [20], Cover [2]) is an example of ranking/unranking algorithms that are well studied in the combinatorial literature (Nijenhuis and Wilf [14]). We refer to this as the enumerative approach. The second (Knuth [12]) is a special-purpose, highly efficient technique that works for balanced codes, i.e., when \( w = \lfloor (n/2) \rfloor \), and is referred to as the complementation method.

The enumerative approach orders the codewords lexicographically (with respect to the partial order defined by \( 0 < 1 \)), as in a dictionary. The encoder computes the codeword from its dictionary index, and the decoder computes the dictionary index from the codeword. The method is effective because there is a simple formula involving binomial coefficients for computing the lexicographic index of a codeword. The resulting code is fully efficient in the sense that \( \eta_1 = 1 \). However, this method requires the computation of the exact values of binomial coefficients \( \binom{n}{k} \), and requires registers of length \( O(n) \), which limits its usefulness.

An alternative is to use arithmetic coding (Rissanen and Langdon [18], Rissanen [17]; see also Cover and Thomas [3, §13.3]). Arithmetic coding is an efficient variable length source coding technique for finite alphabet sources. Given a source alphabet and a simple probability model for sequences \( x \), let \( p(x) \) and \( F(x) \) denote the probability distribution and cumulative distribution function, respectively. An arithmetic encoder represents \( x \) by a number in the interval \( (F(x) - p(x), F(x)) \). The implementation of such a coder can also run into problems with very long registers, but elegant finite-length implementations are known and are widely used (Witten, Neal and Cleary [30]). For constant weight codes, the idea is to reverse the roles of encoder and decoder, i.e., to use an arithmetic decoder as an encoder and an arithmetic encoder as a constant weight decoder (Ramabadran [16]). Ramabadran gives an efficient algorithm based on an adaptive probability model, in the sense that the probability that the incoming bit is 1 depends on the number of 1’s that have already occurred. This approach successfully overcomes the finite-register-length constraints associated with computing the binomial coefficients and the resulting efficiency is often very high, in many cases the loss of information being at most one bit. The encoding complexity of the method is \( O(n) \).

Knuth’s complementation method [12] relies on the key observation that if the bits of a length-\( n \) binary sequence are complemented sequentially, starting from the beginning, there must be a point at which the weight is equal to \( \lfloor n/2 \rfloor \). Given the transformed sequence, it is possible to recover the original sequence by specifying how many bits were complemented (or the weight of the original sequence). This information is provided by a (relatively short) constant weight check string, and the resulting code consists of the transformed sequence followed by the constant weight check bits. In a series of papers, Bose and colleagues extended Knuth’s method in various ways, and determined the limits of this approach (see [31] and references therein). The method is simple and efficient, and even though the overall complexity is \( O(n) \), for \( n = 100 \) we found it to be eight times as fast as the method based on arithmetic codes. However, the method only works for balanced codes, which restricts its applicability.

The two techniques that we have described above both have complexity that depends on the length \( n \) of the codewords. In contrast, the complexity of our algorithm depends only on the weight \( w \), which makes it more suitable for codes with relatively low weight.

As a final piece of background information, we define what we mean by a dissection. We assume the reader is familiar with the terminology of polytopes (see for example Coxeter [4], Grünbaum [8], Ziegler [32]). Two polytopes \( P \) and \( Q \) in \( \mathbb{R}^w \) are said to be congruent if \( Q \) can be obtained from \( P \) by a translation, a rotation and possibly a reflection in a hyperplane. Two polytopes \( P \) and \( Q \) in \( \mathbb{R}^w \) are said to be equidecomposable if they can be decomposed into finite sets of polytopes \( P_1, \ldots, P_t \) and \( Q_1, \ldots, Q_t \), respectively, for some positive integer \( t \), such that \( P_i \) and \( Q_i \) are congruent for all \( i = 1, \ldots, t \) (see Frederickson [7]). That is, \( P \) is the disjoint union of the polytopes \( P_i \), and similarly for \( Q \). If this is the case then we say that \( P \) can be dissected to give \( Q \) (and that \( Q \) can be dissected to give \( P \)).

Note that we allow reflections in the dissection: there are at least four reasons for doing so. (i) It makes no difference.
to the existence of the dissection, since if two polytopes are
equidecomposable using reflections they are also equidecom-
posable without using reflections. This is a classical theorem
in two and three dimensions [7, Chap. 20] and the proof is
easily generalized to higher dimensions. (ii) When studying
congruences, it is simpler not to have to worry about whether
the determinant of the orthogonal matrix has determinant $+1$
or $-1$. (iii) Allowing reflections often reduces the number of
pieces. (iv) Since our dissections are mostly in dimensions
greater than three, the question of “physical realizability” is
usually irrelevant. Note also that we do not require that the
$P_i$
can be obtained from $P$ by a succession of cuts along infinite
hyperplanes. All we require is that $P$ be a disjoint union of
the $P_i$.

One final technical point: when defining dissections using
coordinates, as in Eqns. (3), (4) below, we use a mixture of
$\leq$ and $<$ signs in order to have unambiguously defined maps.
This is essential for our application. On the other hand, it
means that the “pieces” in the dissection may be missing
certain boundaries. It should therefore be understood that if
we were focusing on the dissections themselves, we would
replace each piece by its topological closure.

For further information about dissections see the books
mentioned in Section I.

III. THE GEOMETRIC INTERPRETATION

In this section, we first consider the problem of encoding
and decoding a binary constant weight code of weight $w = 2$
and arbitrary length $n$, i.e., where there are only two bits
set to 1 in any codeword. Our approach is based on the
fact that vectors of weight two can be represented as points
in two-dimensional Euclidean space, and can be scaled, or
normalized, to lie in a right triangle. This approach is
then extended, first to weight $w = 3$, and then to arbitrary weights
$w$.

For any weight $w$ and block length $n$, let $C_w$ denote the
set of all weight $w$ vectors, with $|C_w| = \binom{n}{w}$. Our codebook
$C$ will be a subset of $C_w$, and will be equal to $C_w$ for a
fully efficient code, i.e., when $\eta_1 = 1$. We will represent a
codeword by the $w$-tuple $y := (y_1, y_2, \ldots, y_w)$, $1 \leq y_1 <
y_2 < \ldots < y_w \leq n$, where $y_i$ denotes the position of the $i$th
index in the codeword, counting from the left. If we normalize
these indices $y'_i$ by dividing them by $n$, the codebook
$C$ becomes a discrete subset of the polytope $T_w$, the convex hull
of the points $0^w, 0^w - 1, 0^w - 2, 1, \ldots, 0^w - 1, 1^w$. $T_2$ is a
right triangle, $T_3$ is a right tetrahedron and in general we will call
$T_w$ a unit orthoscheme$^1$.

The set of inputs to the encoder will be denoted by $R_w$:
we assume that this consists of $w$-tuples $y := (y_1, y_2, \ldots, y_w)$
which range over a $w$-dimensional hyper-rectangle or “brick”.
After normalization by dividing the $y_i$ by $n$, we may assume
that the input vector is a point in the hyper-rectangle or “brick”

$$B_w := \{0, 1\} \times \{1 - 2/1, 1\} \times \cdots \times \{1 - 1/w, 1\}.$$

$^1$An orthoscheme is a $w$-dimensional simplex having an edge path consisting
of $w$ totally orthogonal vectors (Coxeter [4]). In a unit orthoscheme these
edges all have length 1. Hadwiger [9] showed that $T_w$ is equidecomposable with
a cube.

We will use $x := (x_1, x_2, \ldots, x_w) = y/n \in B_w$ and
$x'(x'_1, x'_2, \ldots, x'_w) = y'/n \in T_w$ to denote the normal-
ized versions of the input vector and codeword, respectively,
defined by $x_i := y_i/n$ and $x'_i := y'_i/n$ for $i = 1, \ldots, w$.

The basic idea underlying our approach is to find a dissec-
tion of $B_w$ that gives $T_w$. The encoding and decoding
algorithms are obtained by tracking how the points $y$ and $y'$
move during the dissection.

The volume of $B_w$ is $1 \times 1/2 \times 1/3 \times \cdots \times 1/w = 1/w!$. This is also
the volume of $T_w$, as the following argument shows. Classify
the points $x = (x_1, \ldots, x_w)$ in the unit cube $[0, 1]^w$ into $w$
regions according to their order when sorted; the regions are
congruent, so all have volume $1/w!$, and the region where the
$x_i$ are in nondecreasing order is $T_w$.

We now return to the case $w = 2$. There are many ways
to dissect the rectangle $B_2$ into the right triangle $T_2$. We
will consider two such dissections, both two-piece dissections
based on Fig. 2.

In the first dissection, the triangular piece marked 1 in Fig. 2
is rotated clockwise about the center of the square until it
reaches the position shown on the right in Fig. 2. In the second
dissection, the piece marked 1 is first reflected about the main
diagonal of the square and then slid down until it reaches the
position shown on the right in Fig. 2. In both dissections the
code marked 2 is fixed.

The two dissections can be specified in terms of coordi-
ates$^2$ as follows. For the first dissection, we set

$$\begin{array}{ll}
(x_1', x_2') := (x_1, x_2) & \text{if } x_1 < x_2 \\
(x_1', x_2') := (1 - x_1, 1 - x_2) & \text{if } x_1 \geq x_2
\end{array}$$

and for the second, we set

$$\begin{array}{ll}
(x_1', x_2') := (x_1, x_2) & \text{if } x_1 < x_2 \\
(x_1', x_2') := (x_2 - 1/2, x_1 - 1/2) & \text{if } x_1 \geq x_2
\end{array}$$

The first dissection involves only a rotation, but seems
harder to generalize to higher dimensions. The second one
is the one we will generalize; it uses a reflection, but as
mentioned at the end of Section II, this is permitted by the
definition of a dissection.

We next illustrate how these dissections can be converted
into encoding algorithms for constant weight (weight 2) binary

\[1, 2\]

\[B_2\]

\[T_2\]

\[\text{Fig. 2. Two ways to dissect rectangle } B_2 \text{ to give triangle } T_2. \text{ Piece 1 may}
\text{be rotated to its new position, or reflected and translated.}\]
codes. Again there may be several solutions, and the best algorithm may depend on arithmetic properties of \( n \) (such as its parity). We work now with the unnormalized sets \( \mathcal{R}_2 \) and \( \mathcal{C}_2 \). In each case the output is a weight-2 binary vector with 1’s in positions \( y'_1 \) and \( y'_2 \).

**A. First Dissection, Algorithm 1**

1) The input is an information vector \( (y_1, y_2) \in \mathcal{R}_2 \) with \( 1 \leq y_1 \leq n-1 \) and \( \lceil n/2 \rceil + 1 \leq y_2 \leq n \).

2) If \( y_1 < y_2 \), we set \( y'_1 = y_1, \ y'_2 = y_2 \), otherwise we set \( y'_1 = n - y_1 \) and \( y'_2 = n - y_2 + 1 \).

For \( n \) even, this algorithm generates all possible \( n(n-1)/2 \) codewords. For \( n \) odd it generates only \( (n-1)^2/2 \) codewords, leading to a slight inefficiency, and the following algorithm is to be preferred.

**B. First Dissection, Algorithm 2**

1) The input is an information vector \( (y_1, y_2) \in \mathcal{R}_2 \) with \( 1 \leq y_1 \leq n, \lceil (n+1)/2 \rceil + 1 \leq y_2 \leq n \).

2) If \( y_1 < y_2 \), we set \( y'_1 = y_1, \ y'_2 = y_2 \), otherwise we set \( y'_1 = n - y_1 + 1, \ y'_2 = n - y_2 + 2 \).

For \( n \) odd, this algorithm generates all \( n(n-1)/2 \) codewords, but for \( n \) even it generates only \( n(n-1)/2 \) codewords, again leading to a slight inefficiency.

**C. Second Dissection**

1) The input is an information vector \( (y_1, y_2) \in \mathcal{R}_2 \) with \( 1 \leq y_1 \leq n, \lceil n/2 \rceil + 1 \leq y_2 \leq n \).

2) If \( y_1 < y_2 \), we set \( y'_1 = y_1, \ y'_2 = y_2 \), otherwise we set \( y'_1 = y_2 - \lceil n/2 \rceil, \ y'_2 = y_1 - \lceil n/2 \rceil + 1 \).

For \( n \) even, this algorithm generates all \( n(n-1)/2 \) codewords, but for \( n \) odd it generates only \( (n-1)^2/2 \) codewords, leading to a slight inefficiency. There is a similar algorithm, not given here, which is better when \( n \) is odd.

Note that only one test is required in any of the encoding algorithms. The mappings are invertible, with obvious decoding algorithms corresponding to the inverse mappings from \( \mathcal{C}_2 \) to \( \mathcal{R}_2 \).

We now extend this method to weight \( w = 3 \). Fortunately, the Dehn invariants for both the brick \( B_3 \) and our unit orthoscheme \( T_3 \), which is the tetrahedron\(^3\) with vertices \((0, 0, 0), (0, 0, 1), (0, 1, 1)\) and \((1, 1, 1)\), are zero (since in both cases all dihedral angles are rational multiples of \( \pi \), and so by the Dehn-Sydlers theorem the polyhedra \( B_3 \) and \( T_3 \) are equidecomposable. As already mentioned in Section I, the Dehn-Sydlers theorem applies only in three dimensions. But it will follow from the algorithm given in the next section that \( B_w \) and \( T_w \) are equidecomposable in all dimensions.

We will continue to describe the encoding step (the map from \( B_w \) to \( T_w \)) first. We will give an inductive dissection (see Fig. 3), transforming \( B_3 \) to \( T_3 \) in two steps, effectively reducing the dimension by one at each step. In the first step, the brick \( B_3 \) is dissected into a triangular prism (the product of a right triangle, \( T_2 \), and an interval), and in the second step this triangular prism is dissected into the tetrahedron \( T_3 \). Note that the first step has essentially been solved by the dissection given in Eqn. (4).

For the second step we use a four-piece dissection of the triangular prism to the tetrahedron \( T_3 \). This dissection, shown with the tetrahedron and prism superposed in Fig. 4 appears to be new.

There is a well-known dissection of the same pair of polyhedra that was first published by Hill in 1896 [10]. This also uses four pieces, and is discussed in several references: see Boltianskii [1, p. 99], Cromwell [5, p. 47], Frederickson [7, Fig. 20.4], Sydler [23], Wells [29, p. 251]. However, Hill’s dissection seems harder to generalize to higher dimensions. Sydler’s dissection does have the advantage over ours that it can be accomplished purely by translations and rotations, whereas in our dissection two of the pieces (pieces labeled 2 and 3 in Fig. 4) must also be reflected. However, as mentioned at the end of Section II, this is permitted by the definition of a dissection, and is not a drawback for our application. Apart from this, our dissection is simpler than Hill’s, in the sense that his dissection requires a cut along a skew plane \((x_1 - x_3 = 1)\), whereas all our cuts are parallel to coordinate axes.

To obtain the four pieces shown in Fig. 4, we first make two horizontal cuts along the planes \( x_3 = \frac{3}{4} \) and \( x_3 = \frac{2}{3} \), dividing the tetrahedron into three slices. We then cut the middle slice into two by a vertical cut along the plane \( x_2 = \frac{1}{2} \).

There appears to be a tradition in geometry books that discuss dissections of not giving coordinates for the pieces. To an engineer this seems unsatisfactory, and so in Table I we list the vertices of the four pieces in our dissection. Piece 1 has four vertices, while the other three pieces each have

\(^3\)To solve Hilbert’s third problem, Dehn showed that this tetrahedron is not equidecomposable with a regular tetrahedron of the same volume.
six vertices. (In the Hill dissection the numbers of vertices of the four pieces are 4, 5, 6 and 6 respectively.) Given these coordinates, it is not difficult to verify that the four pieces can be reassembled to form the triangular prism, as indicated in Fig. 4. As already remarked, pieces 2 and 3 must be reflected (or “turned over” in a fourth dimension). The correctness of the dissection also follows from the alternative description of this dissection given below.

The dissection shown in Fig. 4 can be described algebraically as follows. We describe it in the more logical detail. The triangular prism is represented by the set of transformations has determinant 

$$-1$$

see from Eqn. (5) that in the second and third cases the linear operation is invertible.

$$w$$

The last column of Fig. 5 shows the ranges of the $$x_i$$ in the four cases; the fact that these ranges are disjoint guarantees that the mapping from $$x_1, x_2, x_3$$ to $$x_1', x_2', x_3'$$ is invertible. The ranges of the $$x_i'$$ will be discussed in more detail in the following section after the general algorithms are presented.

This operation can now be described without explicitly mentioning the underlying dissection. Each interval of length $$1/w$$, together with the given $$x_i$$ values within it, is treated as a single complete unit. In the “cut and paste” operations, these units are rearranged and relabeled in such a way that the operation is invertible.

### IV. Algorithms and Proof of Correctness

In the previous section we provided an encoding and decoding algorithm for weights $$w = 2$$ and $$w = 3$$, based on our geometric interpretation of $$C_2$$ and $$C_3$$ as points in $$\mathbb{R}^w$$. In this section, the algorithm is generalized to larger values of the weight $$w$$. We start with the geometry, and give a dissection of the “brick” $$B_w$$ into the orthoscheme $$T_w$$. We work with the normalized coordinates $$x_i = y_i/n$$ (for a point in $$B_w$$) and $$x_i' = y_i'/n$$ (for a point in $$T_w$$), where $$1 \leq i \leq w$$. Later in this section, we discuss the modifications needed to take into account the fact that the $$y_i'$$ must be integers.

Before we proceed, we make a few remarks. (i) The algorithm is described assuming the source word is represented as a point in the “brick”. In practice, one may need to convert a given source word or index (possibly in its binary representation) to such a representation; this is a straightforward process—we shall return to this at the end of this section. (ii) When calculating the complexity, we assume the computation is performed on operands of length $$O(\log n)$$. (iii) In the complexity analysis, we do not include the cost of setting $$w$$ positions to 1 during encoding, and extracting the positions of the 1’s in the length $$n$$ constant weight codewords during decoding, for two reasons: (a) this is part of the transmission/reception process (or the write/read process), and does not contribute to the overall computation load, and (b) during encoding, setting certain positions to 1 can be done efficiently on a codeword preset to all zeros.

#### A. An Inductive Decomposition of the Orthoscheme

Restating the problem, we wish to find a bijection $$F_w$$ between the sets $$B_w$$ and $$T_w$$. The inductive approach developed for $$w = 3$$ (where the $$w = 2$$ case was a subproblem) will be generalized. Of course the bijection $$F_1$$ between $$B_1$$ and $$T_1$$
is trivial. We assume that a bijection $F_{w-1}$ is known between $B_{w-1}$ and $T_{w-1}$, and show how to construct a bijection $F_w$ between $B_w$ and $T_w$.

The last step in the induction uses a map $f_w$ from the prism $T_{w-1} \times [1 - \frac{1}{w}, 1)$ to $T_w$ ($f_2$ is the map described in Eqn. (4) and $f_3$ is described in Eqn. (5)). The mapping $F_w$ from $B_w$ to $T_w$ is then given recursively by $F_w : (x_1, x_2, \ldots, x_w) \mapsto (x'_1, x'_2, \ldots, x'_w)$, where

$$(x'_1, x'_2, \ldots, x'_w) := f_w(F_{w-1}(x_1, x_2, \ldots, x_{w-1}), x_w).$$

For $w = 1$ we set

$$F_1 := f_1 : B_1 \to T_1, \ (x_1) \mapsto (x'_1) = (x_1).$$

By iterating Eqn. (6), we see that $F_w$ is obtained by successively applying the maps $f_1, f_2, \ldots, f_w$.

The following algorithm defines $f_w$ for $w \geq 2$. We begin with an algebraic definition of the mapping and its inverse, and then discuss it further in the section below. The input to the mapping $f_w$ is a vector $\mathbf{x} := (x_1, x_2, \ldots, x_w)$, with $(x_1, x_2, \ldots, x_{w-1}) \in T_{w-1}$ and $x_w \in [1 - 1/w, 1)$; the output is a vector $\mathbf{x'} := (x'_1, x'_2, \ldots, x'_w) \in T_w$.

**Forward mapping $f_w$ (w ≥ 2):**

1) Let

$$i_0 := \min\{i \in \{1, \ldots, w\} \mid x_w \leq x_i\}.$$

2) Let

$$j_0 := \min\{i \in \{1, \ldots, i_0\} \mid w - i_0 + i - 1 \leq wx_i\} - 1.$$

3) Set $x'_k$ equal to:

$$\begin{align*}
    x'_{k+j_0} - \frac{w + j_0 - i_0}{w} & \quad \text{for } k = 1, \ldots, i_0 - j_0 - 1, \\
    x'_w - \frac{w + j_0 - i_0}{w} & \quad \text{for } k = i_0 - j_0, \\
    x'_{k+j_0-1} - \frac{w + j_0 - i_0}{w} & \quad \text{for } k = i_0 - j_0 + 1, \ldots, w - j_0, \\
    x'_{k-w+j_0} + \frac{i_0 - j_0}{w} & \quad \text{for } k = w - j_0 + 1, \ldots, w.
\end{align*}$$

Eqn. (7) identifies the “cut and paste” operations required to obtain $x'_k$ for different ranges of the variable $k$. If the initial index in one of the four cases in Eqn. (7) is smaller than the final index, that case is to be skipped. A case is also skipped if the subscript for an $x_i$ is not in the range $1, \ldots, w$. Note in Step 1 that $i_0 = w$ if $x_w$ is the largest of the $x_i$'s. This implies that $j_0 = 0$, and then Step 3 is the identity map.

The inverse mapping $g_w$ from $T_w$ to $B_w$ has a similar recursive definition. The $w$th step in the induction is the map $g_w : T_w \to T_{w-1} \times [1 - \frac{1}{w}, 1)$ defined below. For $w = 1$ we set

$$G_1 := g_1 : T_1 \to B_1, \ (x'_1) \mapsto (x_1) = (x'_1).$$

The map $G_w$ is obtained by successively applying the maps $g_w, g_{w-1}, \ldots, g_1$.

**Inverse mapping $g_w$ (w ≥ 2):**

1) Let

$$m_0 := \max\{i \in \{1, \ldots, w\} \mid i - 1 \leq wx'_i\}.$$

2) If $m_0 = w$, let $j_0 := 0$, otherwise let

$$j_0 := w - \max\{i \in \{m_0 + 1, \ldots, w\} \mid wx'_i \leq m_0\};$$

in either case, let $i_0 := j_0 + m_0$.

3) Set $x_k$ equal to:

$$\begin{align*}
    x'_{k+w-j_0} - \frac{i_0 - j_0}{w} & \quad \text{for } k = 1, \ldots, j_0, \\
    x'_{k-j_0} + \frac{w + j_0 - i_0}{w} & \quad \text{for } k = j_0 + 1, \ldots, i_0 - 1, \\
    x'_{k-j_0+1} + \frac{w + j_0 - i_0}{w} & \quad \text{for } k = i_0, \ldots, w - 1, \\
    x'_{k-w+j_0} + \frac{i_0 - j_0}{w} & \quad \text{for } k = w.
\end{align*}$$

Note that the transformations in Eqn. (7) and Eqn. (8) are formal inverses of each other, and that these transformations are volume-preserving. The underlying linear transformations are orthogonal transformations with determinant $+1$ or $-1$.

Before proceeding further, let us verify that in the case $w = 3$, the mapping $f_w = f_3$ agrees with that given in Eqn. (5).

- If $x_1 \leq x_3 < x_2$, then $i_0 = 3$, $j_0 = 0$ and the map is the identity, as mentioned above.
- If $x_1 < x_3 \leq x_2$ there are two subcases:
  - If $\frac{1}{3} \leq x_1$ then $i_0 = 2$, $j_0 = 0$.
  - If $x_1 < \frac{1}{3}$ then $i_0 = 2$, $j_0 = 1$. 

---

**Fig. 5.** Cut-and-paste description of the inverse transformation from triangular prism to tetrahedron.
• If \( x_3 \leq x_1 \leq x_2 \), then \( i_0 = 1, j_0 = 0 \).

The transformations in Eqn. (7) now exactly match those in Eqn. (5).

B. Interpretations and Explanations

In Fig. 6, we give a graphical interpretation of the algorithm, which can be regarded as a generalization of the “cut and paste” description given above. This figure shows the transformation defined by the \( w \times h \) step \( f_w \) in the algorithm. At this step, we begin with a list of \( w-1 \) numbers \( (x_1, x_2, \ldots, x_w-1) \) in increasing order, and a further number \( x_w \) which may be anywhere in the interval \([1 - 1/w, 1]\). This list of \( w \) numbers is plotted in the plane as the set of \( w \) points \( (i, wx_i) \) for \( i = 1, 2, \ldots, w \) (indicated by the solid black circles in Fig. 6). In the first step in the forward algorithm, the augmented list \((x_1, x_2, \ldots, x_w)\) is sorted into increasing order. In the sorted list, \( x_w \) now occupies position \( i_0 \), so the point \((w, wx_w)\) moves to the left, to the new position \((i_0, wx_w)\), and the points \((i, wx_i)\) for \( i = i_0 + 1, \ldots, w - 1 \) move to the right. This is indicated by the arrows in the figure. The new positions of these points are marked by hollow circles.

The point \((i_0, wx_w)\) now lies between the grid points \((i_0, w)\) and \((i_0, w-1)\) (it may coincide with the latter point), since \( x_w \geq 1 - \frac{1}{w} \). We draw the line \( y = x + w - i_0 - 1 \) (shown as the dashed-and-dotted line in Fig. 6). This has unit slope and passes through the points \((i_0, w - 1)\) and \((0, w - i_0 - 1)\). The algorithm then computes \( j_0 + 1 \) to be the smallest index \( i \) for which \( x_i \) is on or above this line. Once \( i_0 \) and \( j_0 \) have been determined, the forward mapping proceeds as follows. The points \((i, wx_i)\) for \( i = 1, \ldots, j_0 \) are shifted to the right of the figure and are moved upwards by the amount \((i_0 - j_0)/w\), their new positions being indicated by crosses in the figure. Finally, the origin is moved to the grid point \((j_0, w - i_0 + j_0)\) and the points are reordered. The \( m_0 := i_0 - j_0 \) points which originally had indices \( j_0 + 1, \ldots, i_0 \) become points \( 1, \ldots, m_0 \) after reordering. In the new coordinates, the final positions of the points lie inside the square region \([1, w]\times[1, w]\). The reader can check that this process is exactly equivalent to the algebraic description of \( f_w \) given above.

To recover \( i_0 \) and \( j_0 \), we first determine the value of \( m_0 := i_0 - j_0 \). This can indeed be done since \( m_0 \) is precisely the index of the largest \( wx_i \) that lies on or above the line \( y = x - 1 \) in the new coordinate system. Note that the position of this line is independent of \( i_0 \) and \( j_0 \) and \((x_1', x_2', \ldots, x_w')\). This works because the points \( wx_1, \ldots, wx_w \) in the original coordinate system, before the origin is shifted, are moved right by \( w \) units and upwards by \( w \) units, so points below the dashed-and-dotted line remain below the line. Furthermore, observe that in the new coordinate system the number of points \((i, wx_i')\) below the line \( y = m_0 \) is equal to \( w - j_0 \). Thus the correct \( i_0 \) and \( j_0 \) values may be recovered, and the inverse mapping can be successfully performed.

The following remarks record two properties of the algorithm that will be used later.

Remark 1: Step 2 of the forward algorithm implies that \( x_{j_0} < \frac{w - i_0 + j_0 - 1}{w} \) and \( x_{j_0 + 1} \geq \frac{w - i_0 + j_0}{w} \). It follows that there is no \( i \) in the range \( 1 \leq i \leq w \) for which

\[
\frac{w - i_0 + j_0 - 1}{w} < x_i < \frac{w - i_0 + j_0}{w}.
\]

Remark 2: The forward algorithm produces a vector \( x' \) whose components satisfy

\[
0 \leq x'_1 \leq \cdots \leq x'_{i_0 + j_0 - 1} \leq \cdots \leq x'_{w - j_0} < \frac{i_0 - j_0}{w}, \tag{9}
\]

\[
\frac{i_0 - j_0}{w} \leq x'_{w - j_0 + 1} \leq x'_{w - j_0 + 2} \leq \cdots \leq x'_w < 1, \tag{10}
\]

and

\[
x'_k < \frac{k - 1}{w}, \quad \text{for } w - j_0 + 1 \leq k \leq w. \tag{11}
\]

Eqsns. (9) and (10) follow from the minimizations in Steps 1 and 2 of the forward algorithm, respectively. The right-hand side of Eqn. (11) expresses the fact, already mentioned, that the first \( j_0 \) points remain below the dotted-and-dashed line after they are shifted.

C. Proof of Correctness

We now give the formal proof that the algorithm is correct. This is simply a matter of collecting together facts that we have already observed.

Theorem 1: For any \( w \geq 1 \), the forward mapping \( f_w \) is a one-to-one mapping from \( T_{w-1} \times [1 - 1/w, 1) \) to \( T_w \) with inverse \( g_w \).

Proof: First, it follows from Remark 2 that, for \( x \in T_{w-1} \times [1 - 1/w, 1) \), \( x' = (x'_1, x'_2, \ldots, x'_w) \) satisfies \( 0 \leq x'_1 \leq \cdots \leq x'_w < 1 \), and so is an element of \( T_w \).

Suppose there were two different choices for \( x \), say \( x^{(1)} \) and \( x^{(2)} \), such that

\[
f_w(x^{(1)}) = f_w(x^{(2)}) = x' \tag{12}.
\]

We know that \( x' \) determines \( m_0, j_0 \) and \( i_0 \). So \( x^{(1)} \) and \( x^{(2)} \) have the same associated values of \( i_0 \) and \( j_0 \). But for a given pair \((i_0, j_0)\), Eqn. (7) is invertible. Hence \( x^{(1)} = x^{(2)} \), and \( f_w \) is one-to-one.

Note that the transformations in Eqn. (7) and Eqn. (8) are inverses of each other. Hence \( f_w \) is also an onto map, and \( g_w \) is its inverse.

D. Number of Pieces

The map \( f_w \), which dissects the prism \( T_{w-1} \times [1 - 1/w, 1) \) to give the orthoscheme \( T_w \), has one piece for each pair \((i_0, j_0)\). If \( i_0 = w \) then \( j_0 = 0 \), while if \( 1 \leq i_0 \leq w - 1 \), \( j_0 \) takes all values from 0 to \( i_0 - 1 \). (It is easy to write down an explicit point in the interior of the piece corresponding to a specified pair of values of \( i_0 \) and \( j_0 \). Assume \( i_0 < w \) and set \( \delta = 1/w^3 \).

Take the point with coordinates \((x_1, \ldots, x_w)\) given by \( x_w = (w - 1)/w + \delta; x_1 = x_w + \delta(i - i_0) \) for \( i = i_0 + 1, \ldots, w - 1 \); \( x_i = (i + w - i_0 - 1 - \delta)/w \) for \( i = 1, \ldots, j_0; x_i = (i + w - i_0 - 1 - \delta)/w \) for \( i = j_0 + 1, \ldots, i_0 - 1 \). The total number of pieces in the dissection is therefore

\[
1 + 1 + 2 + 3 + \cdots + (w - 1) = \frac{w^2 - w + 2}{2}.
\]
which is 1, 2, 4, 7, 11, . . . for \( w = 1, 2, 3, 4, 5, \ldots \). This is a well-known sequence, entry A124 in [21], which by coincidence also arises in a different dissection problem: it is the maximal number of pieces into which a circular disk can be cut with \( w - 1 \) straight cuts. For example, with three cuts, a pizza can be cut into a maximum of seven pieces, and this is also the number of pieces in the dissection defined by \( f_4 \).

\[ n - (w - i) - \left\lfloor \frac{n - (w - i)}{i} \right\rfloor + 1 \leq y_i \leq n - (w - i), \]

for \( i = 1, 2, \ldots, w \), and \( T_{\mathbb{N}}^w \) is the set of vectors \((y_1, y_2, \ldots, y_w) \in \mathbb{N}^w \) satisfying

\[ 1 \leq y_1 < y_2 < \cdots < y_w \leq n. \]

Note that usually \( |B_{\mathbb{N}}^w| \leq |T_{\mathbb{N}}^w| \), which entails a loss in rate.

The forward mapping \( f_w \) is now replaced by the map \( f_{\mathbb{N}}^w \), which sends \((y_1, y_2, \ldots, y_w) \) with \((y_1, y_2, \ldots, y_{w-1}) \in T_{\mathbb{N}}^{w-1} \) and \( n - \left\lfloor \frac{n}{w} \right\rfloor + 1 \leq y_w \leq n \) to an element of \( T_{\mathbb{N}}^w \). Let us write \( n = pw + q \), where \( p \geq 0 \) and \( 0 \leq q \leq w - 1 \). We partition the range 1, 2, \ldots, \( n \) into \( w \) parts, where the first \( n - w - 1 \) parts each have \( p \) elements, the next \( q \) parts each have \( p + 1 \) elements, and the last part has \( p \) elements (giving a total of \( n \) elements). This is similar to the real-valued case, where each interval had length 1/\( w \).

1) Let

\[ i_0 := \min \{ i \in \{1, \ldots, w\} \mid y_w \leq y_i \}. \]
where each short binary sequence to a pair of integers, as in the case of a weight two constant weight codes.

Through proper pairing of the ranges, the loss in the rate can be minimized.

The overall rate loss has two components, the first from the rounding involved in using natural numbers, the second from the loss in the above simplified translation step. However, when the weight is on the order of \( \sqrt{n} \), and \( n \) is in the range of \( 100 \leq 1000 \), the rate loss is usually 1–3 bits per block. For example, when \( n = 529 \), \( w = 23 \), then the rate loss is 2 bits/block compared to the best possible code which would encode \( k_0 = 132 \) information bits.

V. Conclusion

We propose a novel algorithm for encoding and decoding constant weight binary codes, based on dissecting the polytope defined by the set of all binary words of length \( n \) and weight \( w \), and reassembling the pieces to form a hyper-rectangle corresponding to the input data. The algorithm has a natural recursive structure, which enables us to give an inductive proof of its correctness. The proposed algorithm has complexity \( O(w^2) \), independent of the length of the codewords \( n \). It is especially suitable for constant weight codes of low weight.

\[
\begin{align*}
2) \text{Let } j_0 & := \min\{i \in \{1, \ldots, i_0\} \mid V_i < y_i \} - 1, \\
& \text{where } V_i := (w - i_0 + i - 1)p + \max\{q - i_0 + i, 0\}.
\end{align*}
\]

3) Set \( y'_k \) equal to:

\[
\begin{align*}
y_{k+j_0} - V_{j_0+1} & \text{ for } k = 1, \ldots, i_0 - j_0 - 1, \\
y_{w} - V_{j_0+1} & \text{ for } k = i_0 - j_0, \\
y_{k+j_0} - 1 + V_{j_0+1} & \text{ for } k = i_0 - j_0 + 1, \ldots, w - j_0, \\
y_{k+j_0} - n - V_{j_0+1} & \text{ for } k = w - j_0 + 1, \ldots, w.
\end{align*}
\]

The inverse mapping \( g_{w} \) is similarly replaced by the map \( g_{w}^N : T_{w}^N \to \{(y_1, y_2, \ldots, y_w) : (y_1, y_2, \ldots, y_{w-1}) \in T_{w-1}^N, n - \left\lceil \frac{y_w}{w} \right\rceil + 1 \leq y_w \leq n\} \), defined as follows. Again, assume \( n = pw + q \).

1) Let

\[ m_0 := \max\{i \in \{1, \ldots, w\} \mid W_i < y'_i \}, \]

where \( W_i := q + (i - 1)p + \max\{i - q - 1, 0\} \).

2) If \( m_0 = w \), let \( j_0 := 0 \), otherwise let

\[ j_0 := w - \max\{i \in \{m_0 + 1, \ldots, w\} \mid y'_i \leq W_{m_0} + p\}; \]

in either case, let \( i_0 := j_0 + m_0 \).

3) Set \( y'_k \) equal to:

\[
\begin{align*}
y'_{k+w-j_0} - p - W_{m_0} & \text{ for } k = 1, \ldots, j_0, \\
y'_{k+j_0} - n - p - W_{m_0} & \text{ for } k = j_0 + 1, \ldots, i_0 - 1, \\
y'_{k+j_0} - 1 + n - p - W_{m_0} & \text{ for } k = i_0, \ldots, w - 1, \\
y'_{k} - j_0 + n - p - W_{m_0} & \text{ for } k = w.
\end{align*}
\]

We omit the proofs, since they are similar to those for the real-valued case.

F. Comments on the Algorithm

The overall complexity of the transform algorithm is \( O(w^2) \), because at each induction step the complexity is linear in the weight at that step. Recall that the complexities of the arithmetic coding method and Knuth’s complementation method are both \( O(n) \). Thus when the weight \( w \) is larger than \( \sqrt{n} \), the geometric approach is less competitive. When the weight is low, the proposed geometric technique is more efficient, because Knuth’s complementation method is not applicable, while the dissection operations of the proposed algorithm makes it faster than the arithmetic coding method. Furthermore, due to the structure of the algorithm, it is possible to parallelize part of the computation within each induction step to further reduce the computation time.

So far little has been said about mapping a binary sequence to an integer sequence \( y_1, y_2, \ldots, y_w \) such that \( y_i \in [L_i, U_i] \), where \( L_i \) and \( U_i \) are the lower and upper bound of the valid range as specified by the algorithm. A straightforward method is to treat the binary sequence as an integer number and then use “quotient and remainder” method to find such a mapping. However, this requires a division operation, and when the binary sequence is long, the computation is not very efficient. A simplification is to partition the binary sequence into short sequences, and map each short binary sequence to a pair of integers, as in the case of a weight two constant weight codes.

\[ \text{References} \]


