Permutations and Derangements

Tomasz Brengos
Committers : Aliaksei Kudzelka

1 Encoding and Decoding Functions

Let p be a prime number. We work over the finite field $\mathbb{Z}_p$ (the integers mod p). For any positive integer n, the set $\mathbb{Z}_p^n$ denotes the set of all n-tuples of elements from $\mathbb{Z}_p$. Note that since $\mathbb{Z}_p$ is a field, the spaces $\mathbb{Z}_p^n$ and $\mathbb{Z}_p^k$ can be viewed as vector spaces (of dimension n or k respectively) over $\mathbb{Z}_p$.

Encoding function: A map $\xi : \mathbb{Z}_p^k \to \mathbb{Z}_p^n$ is called an *encoding function* (or *coding function*) if it is one-to-one. In other words, ξ takes each possible message (an element of $\mathbb{Z}_p^k$) and encodes it as a longer string (an element of $\mathbb{Z}_p^n$). The image of ξ, denoted

$$\mathcal{C} = \xi(\mathbb{Z}_p^k) \subseteq \mathbb{Z}_p^n,$$

is the set of all possible encoded messages and is called the *code*. The elements of $\mathcal{C}$ are called *codewords*.We usually require ξ to be injective (distinct messages have distinct codewords). Often ξ is a linear transformation (i.e. $\xi(\mathbf{u}+\mathbf{v}) = \xi(\mathbf{u}) + \xi(\mathbf{v})$ and $\xi(a\mathbf{u}) = a\,\xi(\mathbf{u})$ for $a \in \mathbb{Z}_p$), in which case $\mathcal{C}$ is a linear subspace of $\mathbb{Z}_p^n$ of dimension k (such codes are called *linear codes*). Working over the field $\mathbb{Z}_p$ is crucial here, because it ensures we can use the tools of linear algebra and arithmetic modulo p when designing and analyzing codes.

Decoding function: Given an encoding ξ, a map $\eta : \mathbb{Z}_p^n \to \mathbb{Z}_p^k$ is called a *decoding function* for ξ if $\eta(\xi(u)) = u$ for every $u \in \mathbb{Z}_p^k$. In other words, η is a (left) inverse of ξ on the set of codewords, so that decoding an encoded message returns the original message (assuming no errors in transmission).

Once a message $u \in \mathbb{Z}_p^k$ is encoded as a codeword $c = \xi(u) \in \mathbb{Z}_p^n$, it is sent through a channel (transmission). During transmission, some components of c might be altered due to noise or other errors.

2 Error Vectors and Transmission Errors

When a codeword is transmitted (over a noisy channel, or stored and later retrieved), some of its components may change due to errors. We can model the effect of errors by an *error vector*. An *error vector* $\mathbf{e} \in \mathbb{Z}_p^n$ is a vector that, when added to the original codeword, yields the *received word*. If $\mathbf{c} \in \mathcal{C}$ is the sent codeword and $\mathbf{e}$ is the error vector, then the received word is

$$\mathbf{r} = \mathbf{c} + \mathbf{e},$$

where addition is component-wise in $\mathbb{Z}_p$. (In the case $p = 2$, this addition is XOR of bits.) The nonzero entries of $\mathbf{e}$ indicate positions where an error occurred (and the value indicates what was added at that position, e.g. a flip from 0 to 1 in binary is represented by adding 1 mod 2).

The number of errors that occur is the number of nonzero entries in $\mathbf{e}$. This is called the *weight* of $\mathbf{e}$, often denoted $w(\mathbf{e})$. If $w(\mathbf{e}) = t$, we say t errors occurred. The coding and decoding process can be summarized as:

$$\mathbf{u} \xrightarrow{\xi} \mathbf{c} \xrightarrow{\text{error } \mathbf{e}} \mathbf{r} = \mathbf{c} + \mathbf{e} \xrightarrow{\eta} \hat{\mathbf{u}},$$

where $\hat{\mathbf{u}} = \eta(\mathbf{r})$ is the decoder's estimate of the original message. We desire $\hat{\mathbf{u}} = \mathbf{u}$ even if $\mathbf{e}$ is nonzero (up to a certain weight). The design of $\mathcal{C}$ and η determines how many errors can be reliably detected or corrected.

3 Examples of Encoding Functions (Codes)

We now present a few example encoding functions and their corresponding codes:

Example 1: Repetition Code

One simple encoding is the *repetition code*. Here the message space is $\mathbb{Z}_p$ (messages of length 1). The encoding function $\xi_1 : \mathbb{Z}_p \to \mathbb{Z}_p^n$ for some chosen length n is defined by

$$\xi_1(a) = (\underbrace{a, a, \ldots, a}_{n \text{ times}}),$$

i.e. the single-symbol message a is repeated n times to form the codeword. The code $\mathcal{C}_1 = \{(a, a, \ldots, a) : a \in \mathbb{Z}_p\}$ consists of all n-tuples with identical entries. This encoding is one-to-one (different a give different constant tuples).

For example, over $\mathbb{Z}_2$ (the binary case), if $a = 1$ and $n = 5$, the codeword would be $(1, 1, 1, 1, 1)$. If a transmission error flips one of these bits (say we receive $(1, 1, 1, 0, 1)$), the decoder can notice that this is not a valid codeword in $\mathcal{C}_1$ (since not all bits are the same), hence an error is detected. In fact, the repetition code is able to detect up to $n - 1$ errors (any change in at most $n - 1$ positions will yield a tuple that is not constant and thus not in the code). This code even has error-correcting capability: intuitively, the decoder could guess that the majority value is the intended bit (for example, from $(1, 1, 1, 0, 1)$ a reasonable decoded value for a would be 1, since most bits are 1). However, our focus here is on error detection criteria, which we formalize later.

Example 2: Parity Check Code

Another useful encoding adds a *parity check* to the message. For this example, let's work in $\mathbb{Z}_p$ (and especially consider $p = 2$ for binary parity). Let the message space be $\mathbb{Z}_p^{k-1}$ (messages of length $k - 1$). Define an encoding function $\xi_2 : \mathbb{Z}_p^{k-1} \to \mathbb{Z}_p^k$ by

$$\xi_2(a_1, a_2, \ldots, a_{k-1}) = \big(a_1, a_2, \ldots, a_{k-1},\ a_k\big),$$

where the last component a_k is chosen such that the total sum of the k components is 0 in $\mathbb{Z}_p$. In other words,

$$a_k \ = \ -(a_1 + a_2 + \cdots + a_{k-1}) \pmod{p},$$

which ensures $a_1 + a_2 + \cdots + a_{k-1} + a_k \equiv 0 \pmod{p}$. This extra symbol a_k is a redundancy (the *parity symbol*) chosen to enforce a constraint on every codeword.

The resulting code is

$$\mathcal{C}_2 = \{(a_1, \ldots, a_{k-1}, a_k) \in \mathbb{Z}_p^k : a_1 + \cdots + a_{k-1} + a_k = 0\}.$$

In the special case $p = 2$ (binary), this encoding is the standard single-parity-bit code: the last bit a_k is 0 or 1 chosen to make the total number of 1's in the codeword even. For example, if the message is $(1, 0, 1, 1)$ in $\mathbb{Z}_2^4$ (so $k-1 = 4$), then the parity bit a_5 is chosen so that $1+0+1+1+a_5 \equiv 0 \pmod 2$. Here $1+0+1+1 = 3 \equiv 1 \pmod 2$, so we must take $a_5 = 1$ to make the sum even. The encoded 5-bit codeword is $(1, 0, 1, 1, 1)$, which has an even number of 1's (four 1's in this case).

This parity check code allows detection of any single-error in transmission: if one bit of a codeword is flipped, the parity (sum mod p) will no longer be zero, so the received word will violate the code constraint and thus be recognized as invalid. For instance, if $(1, 0, 1, 1, 1)$ is sent and one bit flips, the sum of bits will be odd, alerting us to the error. In fact, more generally, if an odd number of bits are in error, the parity check will detect it (sum becomes nonzero mod 2). However, if an even number of bits are corrupted in the binary case, the parity check could fail to detect it (since an even number of flips can restore the sum to even). We will quantify the error-detecting capability of codes like this in Section 4.

Example 3: PESEL Check Digit

The Polish national identification number, PESEL, consists of 11 decimal digits $p_1p_2\ldots p_{11}$. Its structure is

$$p_1p_2p_3p_4p_5p_6\,p_7p_8p_9p_{10}p_{11}$$

where

- p_1p_2 encode the last two digits of the birth year,
- p_3p_4 encode the month (with a century–dependent offset),
- p_5p_6 encode the day of the month, and
- p_{11} is a *check digit* chosen for error detection.

Formally we regard the first ten digits $(p_1,\ldots,p_{10})\in\mathbb{Z}_{10}^{10}$ as the message and define an encoding

$$\xi_3:\mathbb{Z}_{10}^{10}\longrightarrow\mathbb{Z}_{10}^{11},\qquad \xi_3(p_1,\ldots,p_{10})=(p_1,\ldots,p_{10},p_{11}),$$

where p_{11} is computed via the weighted checksum rule

$$p_{11}\equiv-\big(1p_1+3p_2+7p_3+9p_4+1p_5+3p_6+7p_7+9p_8+1p_9+3p_{10}\big)\pmod{10}.$$

Equivalently, the weighted sum

$$S=1p_1+3p_2+7p_3+9p_4+1p_5+3p_6+7p_7+9p_8+1p_9+3p_{10}+1p_{11}$$

must be a multiple of 10.

For concreteness, take the birth date 15 April 1997, which encodes as 150497 for the first six digits (day 15, month 04, year 97). Suppose the remaining four data digits are 1478, giving the 10–tuple $(1,5,0,4,9,7,1,4,7,8)$. The checksum is

$$\begin{aligned}S&=1\cdot1+3\cdot5+7\cdot0+9\cdot4+1\cdot9+3\cdot7+7\cdot1+9\cdot4+1\cdot7+3\cdot8\\&=1+15+0+36+9+21+7+36+7+24=156\equiv6\pmod{10}.\end{aligned}$$

Hence $p_{11}\equiv-6\equiv4\pmod{10}$, so the full PESEL number is `15049714784`. If any single digit is altered (or two adjacent digits are swapped), the weighted sum is almost certain to be non–zero mod 10, flagging an error. The PESEL checksum therefore detects *all* single–digit errors and the vast majority of simple transpositions.

4 Metric Spaces and Other Distance Metrics

The pair $(\mathbb{Z}_p^n,d_H)$ is an example of a *metric space*. In general, a **metric space** is a set X equipped with a distance function $d:X\times X\to\mathbb{R}$ (called a *metric*) satisfying the three properties listed above (non-negativity and identity of indiscernibles, symmetry, and triangle inequality). The metric $d(x,y)$ represents the "distance" between points x and y in the space.

There are many examples of metrics aside from the Hamming distance. A few common metrics are:

- **Euclidean distance:** For $X=\mathbb{R}^n$, the usual Euclidean metric is

$$d_E((x_1,\ldots,x_n),(y_1,\ldots,y_n))=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2}\,,$$

the standard distance formula derived from the Pythagorean theorem. This metric defines the ordinary geometry of n-dimensional Euclidean space.

- **Maximum (Chebyshev) distance:** For $X=\mathbb{R}^n$ (or $\mathbb{Z}^n$), the maximum metric is defined by

$$d_\infty((x_1,\ldots,x_n),(y_1,\ldots,y_n))=\max_{1\le i\le n}|x_i-y_i|\,.$$

This distance is sometimes called the ℓ_∞ norm or Chebyshev distance. Intuitively, it measures distance in terms of the largest single-coordinate difference. For example, in the plane $\mathbb{R}^2$, the distance between (x_1, y_1) and (x_2, y_2) is $\max(|x_2 - x_1|,\ |y_2 - y_1|)$, which in effect produces a square-shaped notion of distance (all points with d_∞ distance $\leq r$ from a center form a square of side length $2r$).

Many other metrics exist (Manhattan distance, discrete metric, etc.), but the Hamming distance is particularly useful for analyzing codes.

5 Metric Balls and Error Correction

Given a metric space (X, d), a **metric ball** (or *distance ball*) of radius r centered at a point $x \in X$ is the set

$$B_r(x) = \{\, y \in X : d(x, y) \leq r \,\}.$$

It contains all points that are at distance at most r from x. In the Hamming metric space $(\mathbb{Z}_p^n, d_H)$, the ball of radius r around a vector $\mathbf{x}$ consists of all vectors that differ from $\mathbf{x}$ in at most r coordinate positions. For example, in $(\mathbb{Z}_2)^3$ (3-bit binary space),

$$B_1(000) = \{000,\ 001,\ 010,\ 100\},$$

since these are exactly the 3-bit vectors with at most 1 bit different from 000. The size of $B_r(x)$ in a Hamming space depends on r and n; for instance, in $(\mathbb{Z}_2)^3$, a radius-1 ball has $1 + \binom{3}{1} = 4$ elements (including the center).

In coding theory, metric balls are a useful way to visualize and implement error correction. Suppose we have a code $C \subseteq \mathbb{Z}_p^n$. To *correct up to t errors*, the decoding process can be viewed as follows: given a received word $\mathbf{r}$, find the codeword $\mathbf{c} \in C$ that is closest to $\mathbf{r}$ in Hamming distance (i.e. with minimal $d_H(\mathbf{c}, \mathbf{r})$). If the number of errors is at most t, and if certain distance conditions hold, this closest codeword will be the one that was originally sent. In essence, we imagine around each codeword $\mathbf{c}$ a *ball of radius t*. If no two codewords' radius-t balls overlap, then any received word that lies within distance t of a codeword $\mathbf{c}$ could only have come from $\mathbf{c}$ (because if it were also within t of a different codeword $\mathbf{c}'$, then the balls would intersect at $\mathbf{r}$). In that case, we can unambiguously decode $\mathbf{r}$ to $\mathbf{c}$. On the other hand, if the balls of radius t around codewords overlap or touch, a received word could be close to two different codewords, causing ambiguity in decoding.

Metric balls also give a criterion for *detecting* errors (even if we cannot correct them). A code can *detect* up to t errors if whenever up to t errors occur, the result is *not* a valid codeword (unless no error occurred). In terms of distance, this means that if you take any codeword $\mathbf{c}$ and look at all vectors within distance t of $\mathbf{c}$ (excluding $\mathbf{c}$ itself), none of those vectors are codewords in C. Equivalently, the balls of radius t around each codeword $\mathbf{c} \in C$, *not counting the center*, do not contain any other codeword. (They may overlap with balls around other codewords, but only at non-codeword points, which corresponds to errors causing an invalid sequence that the receiver can recognize as erroneous.)

6 A Geometric Example: $(\mathbb{Z}_2)^3$ as a Hamming Space

To visualize these concepts, consider the 3-dimensional binary vector space $(\mathbb{Z}_2)^3$ with the Hamming metric. This space consists of 8 vectors (from 000 to 111). We can geometrically represent $(\mathbb{Z}_2)^3$ as the vertices of a cube, where edges connect vectors that differ in exactly one coordinate (Hamming distance 1 apart). In the figure below, each vertex is labeled by the corresponding 3-bit vector, and each edge represents a single-bit flip. This graph is often called the 3-dimensional *Hamming cube*.

In this cube, one can see how clusters of vectors can be grouped around a given vertex. For instance, as noted, the set $\{000, 001, 010, 100\}$ are all within distance 1 of 000 (a Hamming ball of radius 1 around 000). If we take a simple code such as $C = \{000, 111\}$ (which is the $((3, 1))$ repetition code in binary with $n = 3$, $k = 1$), those codewords correspond to the vertices 000 and 111, which are opposite corners of the

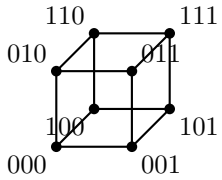

Figure 1: Each vertex corresponds to a 3-bit binary vector. Edges connect vertices that differ in one coordinate (Hamming distance 1). For example, the neighbors of 000 (connected directly by an edge) are 001, 010, and 100, which illustrates all vectors at Hamming distance 1 from 000.

cube. The minimum distance of this code is $d_C = d_H(000, 111) = 3$. The balls of radius 1 around 000 and around 111 (the sets of neighbors each has) do not overlap; in fact, they are separated by at least one bit flip difference. Thus, if at most 1 error occurs during transmission, the received word will remain in the unique "sphere" of radius 1 around whichever codeword was sent, and decoding can correctly deduce the original codeword. If 2 errors occur, the received word will be distance 2 from the original codeword — in this case (for $C = \{000, 111\}$) a double error could yield a vector like 110, which is not a codeword but is now only distance 1 away from the *other* codeword 111. The decoder might misidentify it if it only considers radius 1 spheres, but it will at least notice an error occurred since 110 is not itself a codeword. These ideas will be formalized next.