ENGR 76: Information Science and Engineering Winter 2026 — Lectures 13–15

Error Correcting Codes

Siddharth Chandak

1 Introduction

In the previous module we studied how digital information is transmitted over communication systems, focusing on how binary data is converted into physical signals or waveforms that propagate through a communication channel. Ideally, the receiver would observe exactly the same waveform that was transmitted and recover the original sequence of bits without error. In practice, however, communication channels are imperfect. As signals travel through the channel they may be affected by noise, interference, attenuation, or distortion, which can cause the receiver to interpret some bits incorrectly. Such discrepancies are known as **transmission errors**. Since reliable communication is essential in modern systems, mechanisms are needed to detect when these errors occur and, where possible, to recover the correct data.

This motivates introducing **redundancy** into transmitted data to improve reliability. Instead of sending only the original information bits, the transmitter adds additional bits derived from the data. These extra bits allow the receiver to check whether errors occurred during transmission and, in some cases, to determine the correct data. We next describe these two problems in more detail.

1.1 Error Detection

The first problem is **error detection**, which involves determining whether one or more errors have occurred during transmission. In error detection schemes, additional check information is transmitted together with the data so that the receiver can verify the integrity of the received message. If the received sequence does not satisfy the required consistency checks, the receiver can conclude that an error has occurred.

Error detection is widely used in practice. For example, internet packets often include a checksum or cyclic redundancy check (CRC) that allows the receiver to verify whether the packet was corrupted during transmission. Similarly, many barcode systems include a check digit computed from the other digits; when the barcode is scanned, the system recomputes this value to detect scanning errors.

1.2 Error Correction

The second problem is **error correction**, which involves determining the original transmitted data despite the presence of errors. Error correction schemes introduce structured redundancy into the transmitted data so that the receiver can not only detect that an error has occurred but also infer the most likely original message and correct the corrupted bits.

Error correction is particularly important when retransmission is difficult or impossible. For example, in satellite or deep-space communication, the long delay between transmitter and receiver

makes retransmission impractical. Error correction is also used in compact discs (CDs), where error-correcting codes allow the player to recover the original data even if scratches or imperfections corrupt some bits during reading.

2 Simplest Scheme: Repetition Codes

We begin with one of the simplest ways to introduce redundancy: **repetition**. Suppose we wish to transmit a message that consists of only two possible symbols, represented by bits 0 and 1. Instead of sending the bit once, we transmit it multiple times. More precisely, if the message bit is $b \in \{0, 1\}$, the transmitter sends a sequence consisting of n copies of that bit:

$$0 \mapsto 00\ldots0 \quad (n \text{ times}), \qquad 1 \mapsto 11\ldots1 \quad (n \text{ times}).$$

At the receiver, the repeated bits are examined to infer the original transmitted value. Since the transmitter sends either a sequence of all 0s or all 1s, the receiver looks at the received bits and decides which value appears most often. This is known as a *majority rule*. For example, if three bits are sent and the receiver observes 011, then two of the bits are 1, so the receiver concludes that the original bit was 1. The redundancy introduced by repeating the bit therefore allows the receiver to recover the correct message even if a small number of bits are flipped during transmission.

Example 1. *Consider the case $n = 3$. The encoding rule is*

$$0 \mapsto 000, \qquad 1 \mapsto 111.$$

Suppose the transmitter sends the bit 1, so the transmitted sequence is 111. If a single bit is flipped during transmission, the receiver might observe one of the following: 011, 101, or 110. In each case, two of the three bits are still equal to 1. Using a majority rule*, the receiver concludes that the original bit was 1. Similarly, if 000 is transmitted and a single bit is flipped, the receiver might observe 100, 010, or 001, and the majority rule correctly recovers 0. Thus, when $n = 3$, the repetition scheme can* ***correct one bit error****.*

Example 2. *Now consider the case $n = 2$. The encoding rule is*

$$0 \mapsto 00, \qquad 1 \mapsto 11.$$

Suppose the transmitter sends 1, so the transmitted sequence is 11. If one bit is flipped during transmission, the receiver might observe 10 or 01. In this situation, the receiver knows that an error must have occurred, since neither 10 nor 01 matches a valid encoded message. However, the receiver cannot determine whether the original message was 00 or 11. Therefore, the scheme can ***detect*** *an error but cannot* ***correct*** *it.*

These examples illustrate that the ability to detect or correct errors depends on how different the valid transmitted sequences are from each other. We formalize this in Section 4. We next study general coding schemes, and define required notation to describe a code.

3 General Schemes and Figures of Merit

To study error detection and correction more systematically, we introduce a general framework for coding schemes. A coding scheme specifies a collection of valid sequences that may be sent through the channel. Any sequence that is not in the code is considered invalid.

Definition 1. *A **code** is a set of binary sequences of fixed length that may be transmitted over a communication channel. The elements of this set are called **codewords**.*

3.1 Encoding and Decoding Process

The transmitter begins with an **information sequence**, which is the sequence of bits that we wish to communicate. An encoder maps this sequence to a longer binary sequence called the **encoded sequence**. This encoded sequence is a codeword in the code. The encoded sequence is transmitted over the communication channel. Because of noise or other impairments, the receiver may observe a different sequence of bits, called the **received sequence**. The receiver applies a decoder to the received sequence. The decoder outputs a **decoded sequence**, which is its estimate of the transmitted codeword. From this, the receiver obtains a **decoded information sequence**, which is the estimate of the original information.

When decoding, the receiver compares the received sequence with the valid codewords and chooses the one that appears most likely to have been transmitted. A natural rule is to choose the codeword that differs from the received sequence in the fewest positions. This is called **minimum distance decoding**.

Example 3 (Full Communication Process (Repetition Code))**.** *Consider the repetition code*

$$0 \mapsto 000, \qquad 1 \mapsto 111.$$

- *Information sequence: 1011*
- *Encoded sequence: 111000111111*

Suppose the received sequence is 101100111100. Then, to decode, divide the received sequence into blocks of 3 bits and map each block to the closest codeword.

- *Received sequence: 101100111100*
- *Decoded sequence: 111000111000*
- *Decoded information sequence: 1010*

3.2 Hamming Distance and Minimum Distance

To analyze decoding more formally, we introduce a notion of distance between binary strings.

Definition 2. *The **Hamming distance** between two binary strings of the same length is the number of positions in which the strings differ.*

Definition 3. *The **minimum distance** of a code is the smallest Hamming distance between any two distinct codewords in the code.*

Example 4. *Consider the repetition code with* $n = 3$*:*

$$C = \{000, 111\}.$$

The Hamming distance between the two codewords is 3*. Thus the minimum distance of this code is* $d_{\min} = 3$*.*

Example 5. *Consider the code*

$$C = \{00000, 00111, 11100, 11011\}.$$

Computing pairwise Hamming distances:

- $d_H(00000, 00111) = 3$
- $d_H(00000, 11100) = 3$
- $d_H(00000, 11011) = 4$
- $d_H(00111, 11100) = 4$
- $d_H(00111, 11011) = 3$
- $d_H(11100, 11011) = 3$

The smallest of these distances is 3*, so the minimum distance of the code is* $d_{\min} = 3$*.*

Alternate Perspective

So far, we have viewed coding as the process of adding redundancy to messages before transmission. An equivalent viewpoint is that a code is simply a subset of all binary strings of a fixed length, where the codewords are chosen to be far apart from each other in Hamming distance. From this perspective, designing a good code amounts to selecting binary strings that are well separated; a larger minimum distance between codewords directly leads to better error-detection and error-correction capabilities.

3.3 Figures of Merit

Several quantities are used to evaluate the performance of a coding scheme.

- n: The number of bits in each codeword (block length).
- k: The number of information bits in each message.
- $M = 2^k$: The number of possible messages (equivalently, the number of codewords in the code).
- $d_{\min}$: The minimum Hamming distance between any two distinct codewords.
- Rate:
$$R = \frac{k}{n}.$$
The rate measures the fraction of transmitted bits that carry information.

A good code typically aims to have a large k (more information), small n (shorter transmissions), large rate R, and large $d_{\min}$ (better error-correction capability). These goals, however, are often in tension with each other.

Example 6 (Repetition Code)**.** *Consider the repetition code $C = \{000, 111\}$. Its parameters are $n = 3$, $k = 1$, $M = 2$, rate $= \frac{1}{3}$, and $d_{\min} = 3$.*

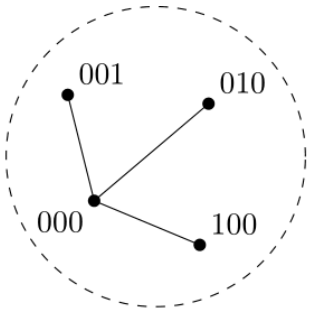
Example 7. *Consider the code $C = \{00000, 00111, 11100, 11011\}$. Its parameters are $n = 5$, $k = 2$, $M = 4$, rate $= \frac{2}{5}$, and $d_{\min} = 3$.*

4 Error Detection and Error Correction Capabilities

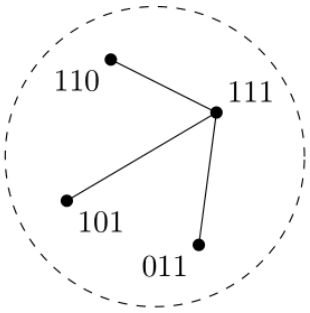
We first define Hamming balls, which will help us both build intuition and formally prove results. For a codeword c and integer $t \geq 0$, define the Hamming ball

$$B(c, t) = \{x : d_H(x, c) \leq t\}.$$

This consists of all binary strings that differ from c in at most t positions. The below figure the Hamming balls of radius 1 around codewords 000 and 111.



noise ball around 000



noise ball around 111

4.1 Error Detection

Let $\mathcal{C}$ be a code with minimum distance $d_{\min}$.

Theorem 1 (Error Detection)**.** *A code can detect up to t bit flips if and only if*

$$d_{\min} \geq t + 1.$$

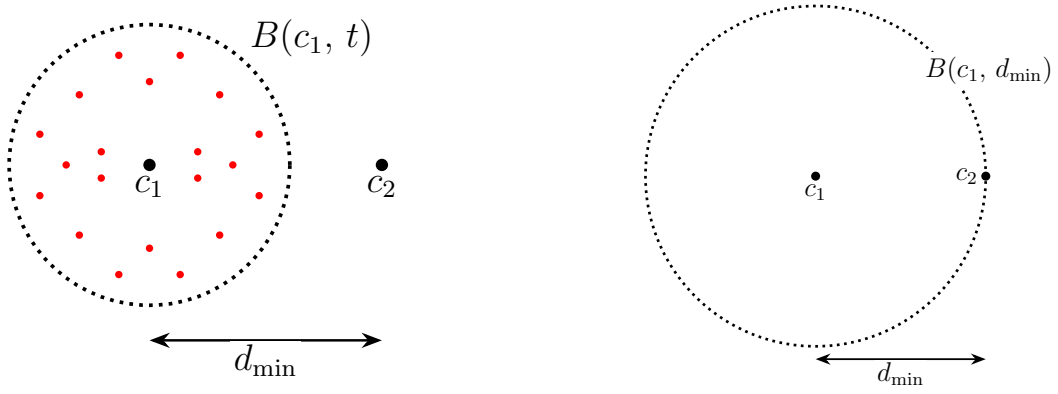
Equivalently, a code with minimum distance $d_{\min}$ can detect up to $d_{\min} - 1$ bit flips.

Proof. Suppose $d_{\min} \geq t + 1$. Let $c_1 \in \mathcal{C}$ be transmitted and suppose at most t bit flips occur. Then the received word r differs from c_1 in at most t positions. Since every other codeword is at distance at least $d_{\min} \geq t + 1$ from c_1, it is impossible for r to equal another codeword. In other words, r cannot coincide with any valid codeword in $\mathcal{C}$. Therefore $r \notin \mathcal{C}$, and the receiver detects that an error has occurred.

Conversely, suppose $d_{\min} \leq t$. Then there exist two distinct codewords c_1 and c_2 whose distance is at most t. If c_1 is transmitted and the channel flips exactly those bits in which c_1 and c_2 differ,

the received word becomes c_2. Since c_2 is also a valid codeword, the receiver has no way to detect that an error occurred. Therefore, the code cannot guarantee detection of t bit flips.

Therefore, the code can detect up to t bit flips if and only if $d_{\min} \geq t + 1$. □



(a) When $t < d_{min}$

(b) When $t = d_{min}$

Geometric Interpretation. The diagrams provide a useful geometric intuition. In the left figure, we draw the Hamming ball $B(c_1, t)$ around c_1. As long as $t < d_{\min}$, this ball does not contain any other codeword such as c_2. Hence any received word inside the ball cannot be another valid codeword, and an error is detected. In the right figure, if we allow $t = d_{\min}$, the ball around c_1 can reach another codeword c_2. In this case, it is possible for c_1 to be transformed into c_2 by $d_{\min}$ bit flips. Since c_2 is itself a valid codeword, the receiver cannot detect that an error occurred. This explains why a code can detect at most $d_{\min} - 1$ bit flips.

4.2 Error Correction

We now consider error correction under minimum-distance decoding.

Theorem 2 (Error Correction)**.** *A code can correct up to t bit flips if and only if*

$$d_{\min} \geq 2t + 1.$$

Equivalently, a code with minimum distance $d_{\min}$ can correct up to

$$\left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

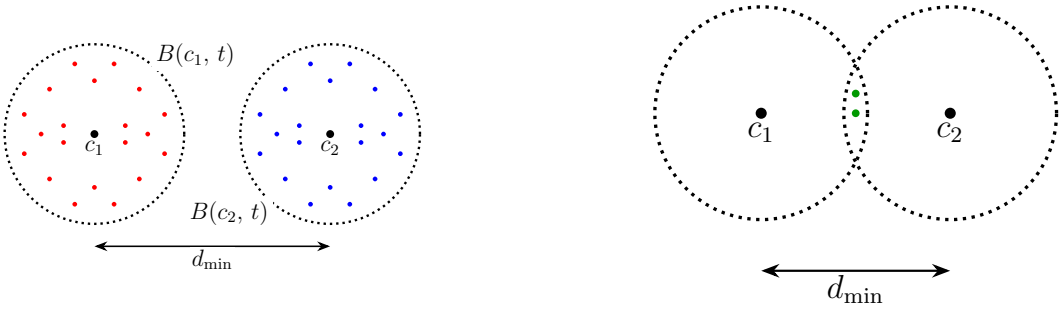
bit flips.

Proof. Suppose first that $d_{\min} \geq 2t+1$. Let c_1 be transmitted and suppose at most t bit flips occur, so that the received word r satisfies $d_H(r, c_1) \leq t$. To decode correctly, we must ensure that r is closer to c_1 than to any other codeword c_2. Since every other codeword satisfies $d_H(c_1, c_2) \geq d_{\min}$, and since r differs from c_1 in at most t positions, the distance from r to c_2 must be at least

$$d_H(c_1, c_2) - d_H(r, c_1) \geq (2t + 1) - t = t + 1.$$

Thus r is strictly closer to c_1 than to any other codeword, and minimum-distance decoding correctly recovers c_1.

Conversely, suppose $d_{\min} \leq 2t$. Then there exist two distinct codewords c_1 and c_2 whose distance is at most $2t$. In this case, the Hamming balls $B(c_1, t)$ and $B(c_2, t)$ overlap. Hence there exists a word r that lies within distance t of both c_1 and c_2. If c_1 is transmitted and r is received, the decoder cannot uniquely determine which codeword was sent. Therefore the code cannot guarantee correction of t bit flips. □



Geometric Interpretation. The figures illustrate the key idea. In the left diagram, the Hamming balls of radius t around distinct codewords do not intersect. Any received word within distance t of a transmitted codeword lies strictly inside its ball and cannot belong to another ball. Thus decoding is unambiguous. In the right diagram, the balls intersect. There exist received words that are within distance t of two different codewords. When this happens, minimum-distance decoding cannot uniquely recover the transmitted codeword. This overlap explains why error correction requires $d_{\min} \geq 2t + 1$.

5 The $(7, 4)$ Hamming Code

Hamming codes form a family of binary linear codes with minimum distance $d_{\min} = 3$. Since a code with minimum distance 3 can correct one bit flip, Hamming codes provide single-error correction with very low decoding complexity. They were the first practical code and were developed in the 1940s by Richard Hamming and are still used in applications such as memory hardware, where fast and simple error correction is required.

In this section, we study the $(7, 4)$ Hamming code, the smallest nontrivial member of this family. It is called a $(7, 4)$ code because each codeword has length $n = 7$ bits, while each message consists of $k = 4$ information bits. The encoder therefore adds three redundancy bits to the four information bits.

5.1 Encoding

Let the information bits be b_1, b_2, b_3, b_4. The encoder produces a 7-bit codeword of the form $b_1b_2b_3b_4p_1p_2p_3$, where the bits p_1, p_2, p_3 are chosen to enforce parity constraints. These parity bits are defined using the XOR operation, denoted by $\oplus$. For two bits, $a \oplus b$ equals 1 if the bits differ

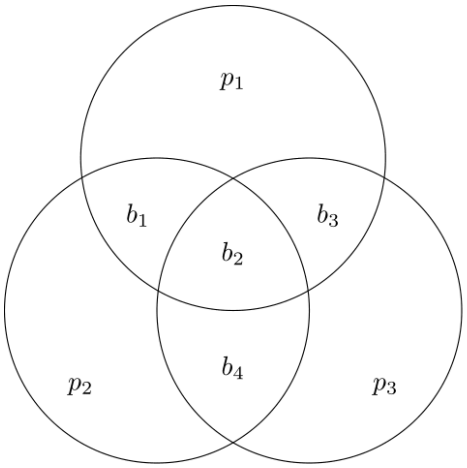
and 0 if they are the same. Equivalently, XOR outputs 1 if an odd number of input bits are 1, and 0 if the number of 1's is even. The parity bits are defined by

$$p_1 = b_1 \oplus b_2 \oplus b_3, \qquad p_2 = b_1 \oplus b_2 \oplus b_4, \qquad p_3 = b_2 \oplus b_3 \oplus b_4.$$

Each parity bit is chosen so that a specific group of bits contains an even number of 1's. For example, the definition of p_1 ensures that the bits b_1, b_2, b_3, and p_1 together contain an even number of 1's. The other parity bits are defined similarly. In this way, every valid codeword satisfies three parity conditions, and each information bit participates in more than one such group. As a result, if a single bit is flipped during transmission, at least one of these conditions is disturbed, allowing the receiver to detect and locate the error.

5.2 Parity-Check Structure

The parity constraints can be visualized using three overlapping parity circles, as shown in the figure. Each circle represents one parity equation. The bits that lie inside a circle are exactly the bits that participate in that parity check. For a valid codeword, the total number of 1's inside each circle must be even.



$$p_1 = b_1 \oplus b_2 \oplus b_3$$
$$p_2 = b_1 \oplus b_2 \oplus b_4$$
$$p_3 = b_2 \oplus b_3 \oplus b_4$$

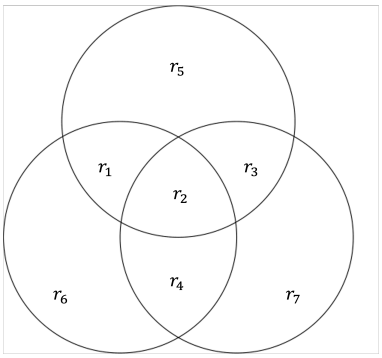
The key structural feature of the diagram is that every bit lies in a unique combination of circles: some bits belong to exactly one circle, some to two circles, and one bit to all three. In particular, no two bits share the same pattern of circle membership. If a single bit flips during transmission, it changes the number of 1's in precisely those circles that contain it. Consequently, the circles whose counts become odd identify exactly which bit was corrupted. This correspondence between circle membership and error location is the fundamental idea behind syndrome decoding in the $(7, 4)$ Hamming code.

5.3 Decoding

Suppose that at most one bit flip occurs during transmission and the receiver observes

$$r_1 r_2 r_3 r_4 r_5 r_6 r_7.$$

To decode, we use the circle diagram shown below.



Each circle groups together certain bits. For a valid codeword, the total number of 1's inside each circle must be even. We therefore count how many 1's appear in each circle of the received word and record whether that count is even or odd. There are four possible situations.

- **All three circles contain an even number of 1's.** In this case, the received word satisfies all three circle conditions. Since flipping a single bit would necessarily change the count in at least one circle, no bit was flipped. For example, consider the received sequence 0101100. All three circles contain an even number of 1's, so no circle has an odd count. Hence no bit was flipped. The decoded information bits are 0101.

- **Exactly one circle contains an odd number of 1's.** The flipped bit must lie in that circle and in no other circle. From the diagram, there is exactly one such bit. Flipping that bit restores the correct counts in all circles. For example, consider the received word 1110110. Only the right circle contains an odd number of 1's. The only bit that lies in the right circle and in no other circle is r_6. Therefore bit 6 was flipped. Flipping r_6 yields the corrected codeword 1110100. The decoded information bits are 1110.

- **Exactly two circles contain an odd number of 1's.** The flipped bit must lie in the overlap of those two circles. The diagram shows that exactly one bit lies in that intersection. Flipping it restores the correct counts. For example, consider the received word 1011101. The top and right circles contain an odd number of 1's. The only bit lying in the intersection of these two circles is r_3. Hence bit 3 was flipped. Flipping r_3 yields the corrected codeword 1001101. The decoded information bits are 1001.

- **All three circles contain an odd number of 1's.** The flipped bit must lie in the central region shared by all three circles. There is exactly one such bit. Flipping it restores the correct counts in every circle. For example, consider the received word 0111110. All three circles contain an odd number of 1's. The flipped bit must therefore lie in the region common to all three circles, which is r_2. Flipping r_2 yields the corrected codeword 0011110. The decoded information bits are 0011.

5.3.1 Formal Syndrome Decoding

The circle-based decoding rule can be written algebraically using the *syndrome bits*. For the received word $r_1r_2r_3r_4r_5r_6r_7$, we compute

$$s_1 = r_1 \oplus r_2 \oplus r_3 \oplus r_5, \qquad s_2 = r_1 \oplus r_2 \oplus r_4 \oplus r_6, \qquad s_3 = r_2 \oplus r_3 \oplus r_4 \oplus r_7.$$

Each s_i equals 1 precisely when the corresponding circle has an odd number of 1's. The vector (s_1, s_2, s_3) is called the syndrome of the received word. If the syndrome equals $(0, 0, 0)$, the received word already satisfies all circle conditions and no error occurred. Otherwise, the nonzero syndrome determines which bit must be flipped. The receiver corrects the error by toggling that bit and then extracts the first four bits as the decoded information sequence.

Is the $(7, 4)$ Hamming Code Optimal? The $(7, 4)$ Hamming code has minimum distance 3 and can therefore correct one bit flip. A natural question is whether a $(7, k)$ code with $k > 4$ could also correct one error, or more generally, how many codewords a length-7 code with minimum distance 3 can contain. In the next section, we address this question using the Hamming bound.

6 Hamming Bounds and the Family of Hamming Codes

6.1 The Hamming Bound for $d_{\min} = 3$

We now derive an upper bound on the size of any binary code capable of correcting one bit flip.

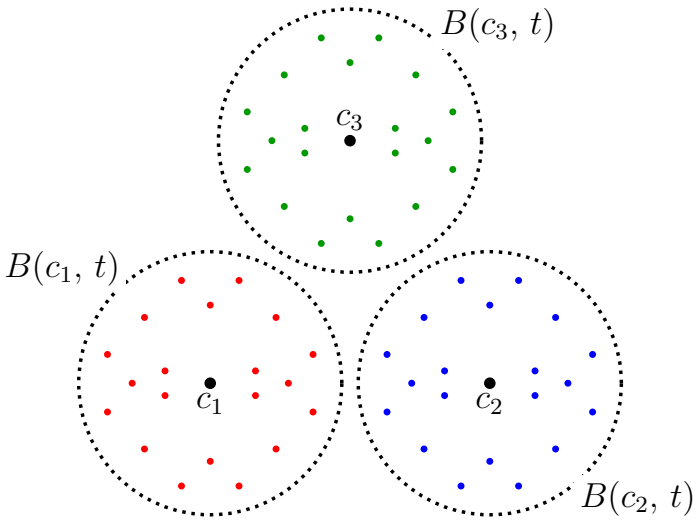
Theorem 3 (Hamming Bound for $d_{\min} = 3$)**.** *Let $\mathcal{C} \subseteq \{0, 1\}^n$ be a binary code of length n with minimum distance $d_{\min} = 3$, and let $M = |\mathcal{C}|$. Then*

$$M \leq \frac{2^n}{1 + n}.$$

Proof. Since $d_{\min} = 3$, the code can correct one bit flip. For each codeword $c \in \mathcal{C}$, consider the Hamming ball

$$B(c, 1) = \{x \in \{0, 1\}^n : d_H(x, c) \leq 1\}.$$

This ball consists of the codeword c itself together with the n words obtained by flipping each coordinate once, so $|B(c, 1)| = 1 + n$.



The figure illustrates these radius-1 Hamming balls around distinct codewords. Because the code corrects one error, a received word must lie in exactly one such ball. Hence the balls $B(c, 1)$ corresponding to distinct codewords must be disjoint.

Suppose, for contradiction, that a word x lies in both $B(c_1, 1)$ and $B(c_2, 1)$ for distinct codewords c_1 and c_2. Then

$$d_H(x, c_1) \leq 1 \quad \text{and} \quad d_H(x, c_2) \leq 1.$$

By the triangle inequality,

$$d_H(c_1, c_2) \leq d_H(c_1, x) + d_H(x, c_2) \leq 2,$$

which contradicts $d_{\min} = 3$. Therefore the balls are pairwise disjoint.

Since all these disjoint balls lie inside $\{0, 1\}^n$, which contains 2^n words, we must have

$$M(1 + n) \leq 2^n.$$

□

A code for which equality holds in the Hamming bound is called a *perfect code*. The Hamming codes form an important family of perfect codes: for appropriate block lengths, they achieve equality in this bound and therefore pack the space $\{0, 1\}^n$ as efficiently as possible while correcting one bit flip. We now describe the general family of Hamming codes.

6.2 The Family of Hamming Codes

We now determine for which parameters equality holds in the Hamming bound. If a code with $d_{\min} = 3$ achieves equality in the Hamming bound, then $M(1 + n) = 2^n$. In particular, $1 + n$ must divide 2^n. Since 2^n is a power of 2, its only positive divisors are powers of 2. Therefore we must have $n + 1 = 2^r$ for some integer $r \geq 1$, so $n = 2^r - 1$. Now suppose that $M = 2^k$. Substituting into the equality condition gives $2^k(1 + n) = 2^n$. Using $1 + n = 2^r$, this becomes $2^k \cdot 2^r = 2^n$, and hence $k + r = n$. Therefore $k = n - r = 2^r - 1 - r$. In general, for each $r \geq 2$, these codes that achieve equality have parameters

$$(n, k, d_{\min}) = \big(2^r - 1,\ 2^r - 1 - r,\ 3\big).$$

These are the *Hamming codes*.

The first few Hamming codes are obtained by choosing small values of r.

- For $r = 2$, we obtain $n = 2^2 - 1 = 3$ and $k = 3 - 2 = 1$. This gives the $(3, 1)$ repetition code (000 and 111).
- For $r = 3$, we obtain $n = 2^3 - 1 = 7$ and $k = 7 - 3 = 4$. This gives the $(7, 4)$ Hamming code.
- For $r = 4$, we obtain $n = 2^4 - 1 = 15$ and $k = 15 - 4 = 11$. This gives the $(15, 11)$ Hamming code.

6.3 The Hamming Bound for General $d_{\min}$

Theorem 4 (General Hamming Bound)**.** *Let $\mathcal{C} \subseteq \{0,1\}^n$ be a binary code of length n with minimum distance $d_{\min}$, and let $M = |\mathcal{C}|$. Define*

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor.$$

Then

$$M \sum_{i=0}^{t} \binom{n}{i} \leq 2^n.$$

Proof. As in the proof of the previous theorem, consider for each codeword $c \in \mathcal{C}$ the Hamming ball

$$B(c,t) = \{x \in \{0,1\}^n : d_H(x,c) \leq t\}.$$

Since $d_{\min} \geq 2t+1$, the code can correct up to t bit flips, and therefore the balls $B(c,t)$ corresponding to distinct codewords must be disjoint. Hence

$$M\,|B(c,t)| \leq 2^n.$$

It remains to compute $|B(c,t)|$, the number of binary strings within distance t of a fixed word c. A word is at Hamming distance exactly i from c if and only if it differs from c in exactly i coordinates. To construct such a word, we choose i positions out of the n coordinates to flip. There are $\binom{n}{i}$ ways to choose these positions, and once the positions are chosen, the resulting word is uniquely determined (since each chosen bit must be flipped). Thus the number of words at distance exactly i from c is $\binom{n}{i}$. Summing over all i from 0 to t, we obtain

$$|B(c,t)| = \sum_{i=0}^{t} \binom{n}{i}.$$

Substituting into the inequality $M\,|B(c,t)| \leq 2^n$ gives

$$M \sum_{i=0}^{t} \binom{n}{i} \leq 2^n.$$

□

This inequality is called the *Hamming bound* or the *sphere-packing bound*. It provides an upper bound on the number of codewords for given n and d_{min}.

7 Convolutional Codes and Encoding

So far, the codes we have studied are **block codes**. In a block code, the information sequence is divided into blocks of a fixed number of bits, and each block is encoded independently into a longer block called a codeword. For example, in the $(7,4)$ Hamming code, every block of four information bits is independently mapped to a block of seven encoded bits. In many communication systems, however, information arrives as a continuous stream of bits rather than as isolated blocks. In

such settings, it is often more natural to encode the sequence sequentially as the bits arrive. This motivates the idea of **convolutional codes**.

A convolutional code is a coding scheme in which the encoded output at each time step depends not only on the current input bit, but also on a fixed number of previous input bits. Thus, unlike block codes where different blocks are encoded independently, in a convolutional code the outputs produced at consecutive time steps are coupled through the encoder memory, so each information bit influences several future encoded bits.

7.1 A Rate-$1/2$ Convolutional Code

We study a specific binary convolutional code whose rate is approximately $1/2$. Let the information sequence be $b_1, b_2, \ldots, b_k$. At each time step j, the encoder produces two output bits $p_j^{(1)}$ and $p_j^{(2)}$ according to

$$p_j^{(1)} = b_j \oplus b_{j-1} \oplus b_{j-2}, \qquad p_j^{(2)} = b_j \oplus b_{j-2}.$$

Thus, each output bit is determined by the current information bit together with the previous two bits. The encoder therefore has memory 2. The complete encoded sequence is $p_1^{(1)}, p_1^{(2)}, p_2^{(1)}, p_2^{(2)}, \ldots, p_k^{(1)}, p_k^{(2)}$. Since each input bit generates two output bits, the rate is $R = \frac{1}{2}$.

7.1.1 Memory and Error Protection

A useful way to understand convolutional encoding is to ask how many encoded bits are affected by a single information bit b_j.

- At time j, the bit b_j directly appears in both outputs:

$$p_j^{(1)} = b_j \oplus b_{j-1} \oplus b_{j-2}, \qquad p_j^{(2)} = b_j \oplus b_{j-2}.$$

- At the next time step, b_j influences only one output bit:

$$p_{j+1}^{(1)} = b_{j+1} \oplus b_j \oplus b_{j-1}.$$

- At time $j+2$, the bit b_j again influences both output bits:

$$p_{j+2}^{(1)} = b_{j+2} \oplus b_{j+1} \oplus b_j, \qquad p_{j+2}^{(2)} = b_{j+2} \oplus b_j.$$

  After this, b_j no longer appears in any future outputs.

Therefore, each information bit affects outputs over three consecutive time steps, and affects five output (encoded) bits.

7.1.2 Initialization and Termination

The encoder equations involve previous bits b_{j-1} and b_{j-2}. At the beginning of transmission, these bits do not yet exist. To resolve this, we use the standard initialization convention

$$b_0 = b_{-1} = 0.$$

With this convention, the first outputs become

$$p_1^{(1)} = b_1, \qquad p_1^{(2)} = b_1,$$

$$p_2^{(1)} = b_2 \oplus b_1, \qquad p_2^{(2)} = b_2.$$

At the end of the information sequence, the final bits would otherwise influence fewer encoded bits than earlier bits. To provide uniform protection, we append two zero bits to the information sequence:

$$b_{k+1} = b_{k+2} = 0.$$

These are called **termination bits** or **tail bits**. They produce four additional encoded bits:

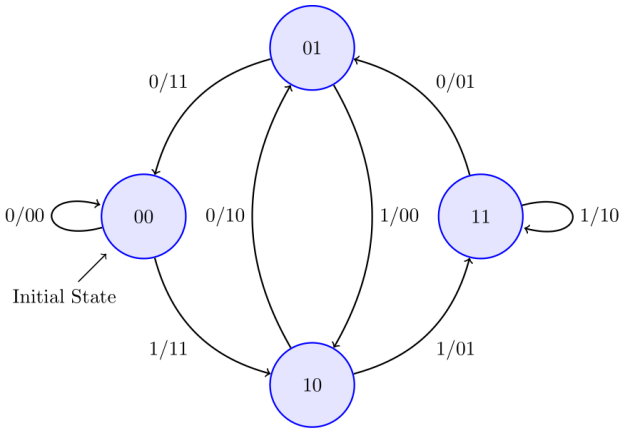
$$p_{k+1}^{(1)} = b_k \oplus b_{k-1}, \qquad p_{k+1}^{(2)} = b_{k-1},$$

$$p_{k+2}^{(1)} = b_k, \qquad p_{k+2}^{(2)} = b_k.$$

Thus the full encoded sequence has length $2(k+2) = 2k+4$. The resulting rate is therefore $R = \frac{k}{2k+4}$, which approaches $1/2$ for large k.

Example 8. *Consider the information sequence* 1011. *Appending two zeros gives* 101100. *It can be verified that the encoded sequence is* 111000010111.

7.2 Finite-State Machine Interpretation



A particularly useful viewpoint is to interpret the encoder as a finite-state machine (FSM). At time j, the encoder only needs to remember the previous two information bits: $b_{j-1}b_{j-2}$. Since each of these bits is binary, there are four possible combinations: 00, 01, 10, 11. These act as the states for our FSM. Each transition corresponds to receiving the current information bit b_j and producing the two encoded bits $p_j^{(1)}, p_j^{(2)}$. The transitions are therefore marked with $b_j/p_j^{(1)}, p_j^{(2)}$. The system always starts in state 00 because of the initialization $b_{-1} = b_0 = 0$. Because we append two zeros at the end, the encoder also terminates in state 00. An interactive visualization showing the transitions for the FSM can be found here.

8 Decoding for Convolutional Codes (Viterbi Decoding)

We now turn to the problem of decoding: given a received bit sequence, our goal is to determine the most likely transmitted information sequence. As before, a natural principle is **minimum distance decoding**, where among all valid encoded sequences we choose the one with the smallest Hamming distance from the received sequence.

8.1 Why Brute-Force Decoding is Difficult

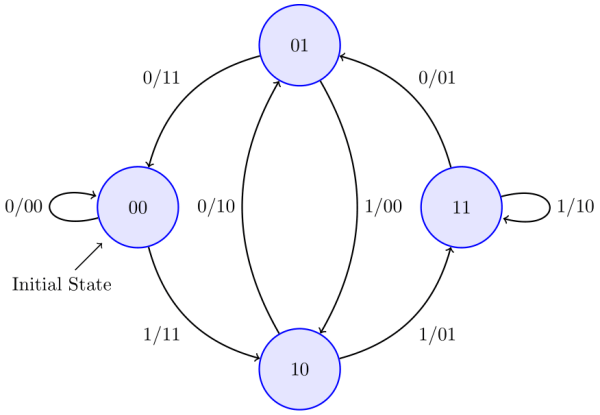
Suppose the information sequence has length k. Since each information bit can be either 0 or 1, there are 2^k possible information sequences. Brute-force minimum distance decoding would require comparing the received sequence with all 2^k possibilities. This is computationally infeasible even for moderate values of k. For example, when $k = 100$, the number of possible sequences is 2^{100}, which is far too large to search exhaustively. We therefore need a more efficient method.

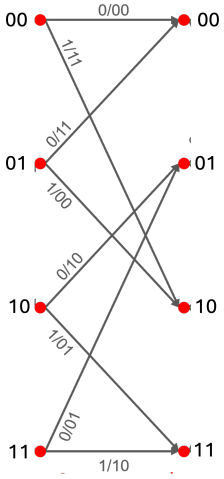
8.2 Decoding Using the State Machine

Suppose that there is **no noise**. In this case, decoding means finding the unique path through the state machine that generates the observed encoded sequence. For example, consider the received sequence 0000110110011100. Starting from the initial state 00, we follow the sequence of transitions whose outputs exactly match the received bits. This allows us to recover the transmitted information sequence. However, if noise is present, an exact match may not exist. For example, suppose the first received pair is 01. Since the initial state is 00, the first transmitted pair could only have been 00 or 11 depending on whether the first information bit was 0 or 1. Thus, based only on the first two received bits, we cannot uniquely determine the first information bit. Instead, we must determine which **entire path** through the state machine is most consistent with the full received sequence. This viewpoint is much easier to visualize and analyze using the trellis representation.

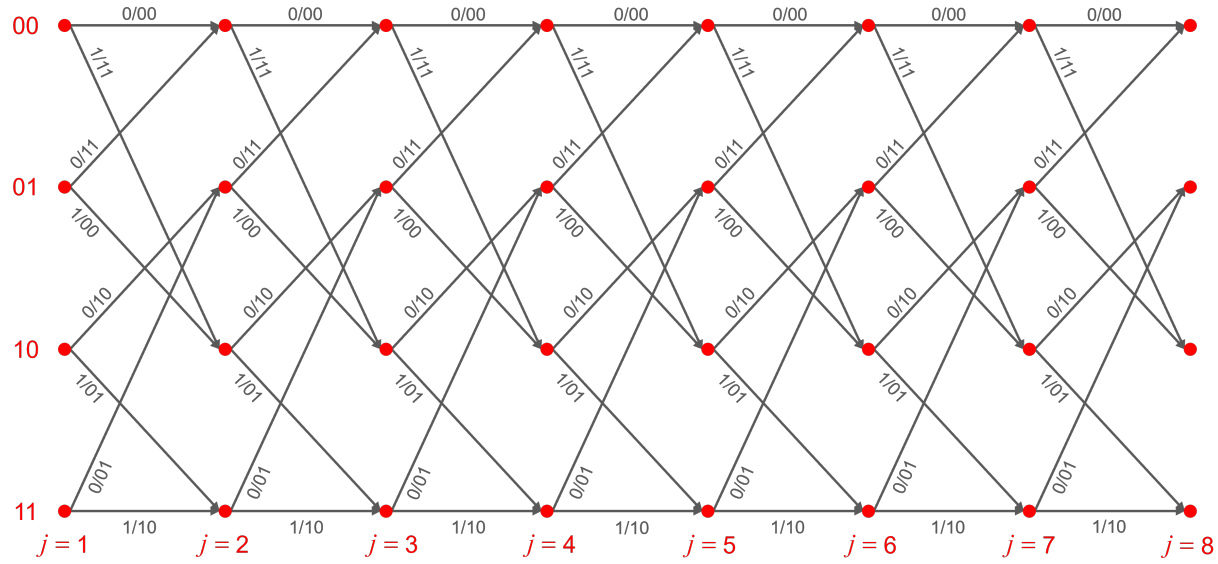
8.3 The Trellis Representation

The **trellis** is obtained by unfolding the finite-state machine across time. Each column corresponds to one time step, and each node in that column represents one of the four possible states.





Extending this over all time steps gives the full trellis.



A complete path through the trellis corresponds exactly to one possible information sequence. Since the encoder starts in state 00 and we append two zeros at the end, every valid path must begin and end at state 00. The trellis figure below gives one example of a valid path through the trellis. The final two zeroes (in green) are the appended tail bits.

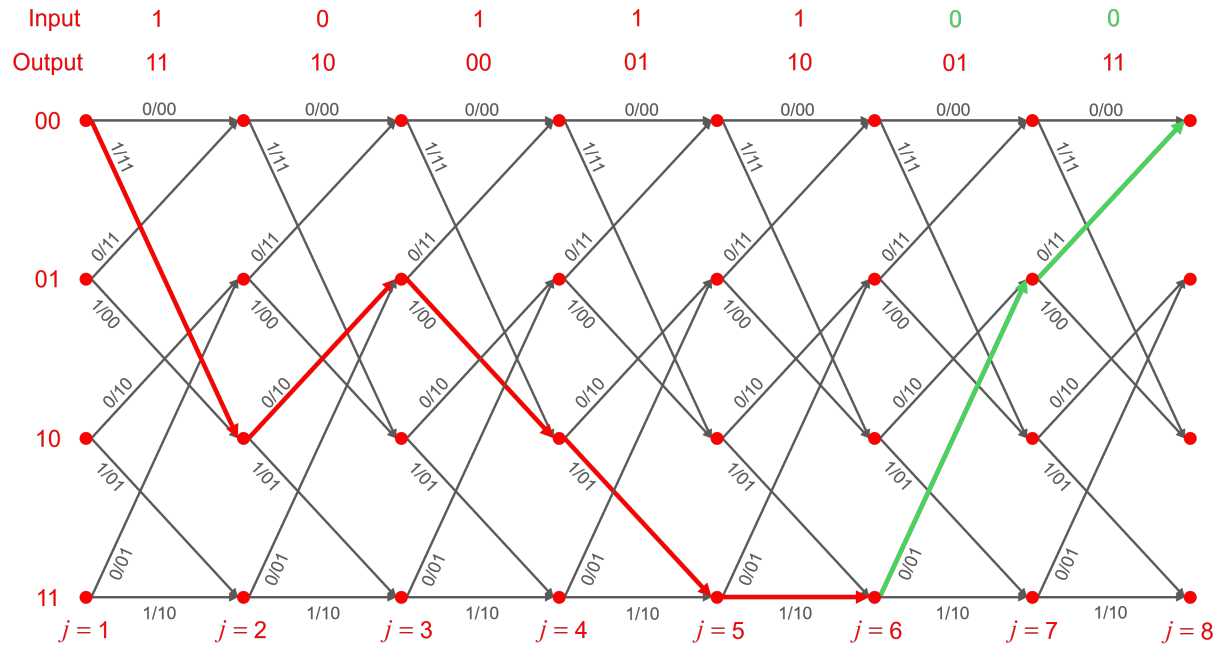


Figure 2: Example of a valid path through the trellis.

8.4 Viterbi Decoding

Decoding reduces to finding the path along the trellis, starting and ending at state 00, that has the minimum Hamming distance from the received sequence. The **Viterbi algorithm** provides an efficient way to solve this problem using **dynamic programming**, by breaking the global decoding problem into a sequence of local decisions made over time. We next describe this algorithm step by step, beginning with the notions of *branch metrics* and *path metrics*.

8.4.1 Branch Metrics

Suppose the received sequence is 01111110000110. We first divide the sequence into pairs of bits:

$$01 \quad 11 \quad 11 \quad 10 \quad 00 \quad 01 \quad 10.$$

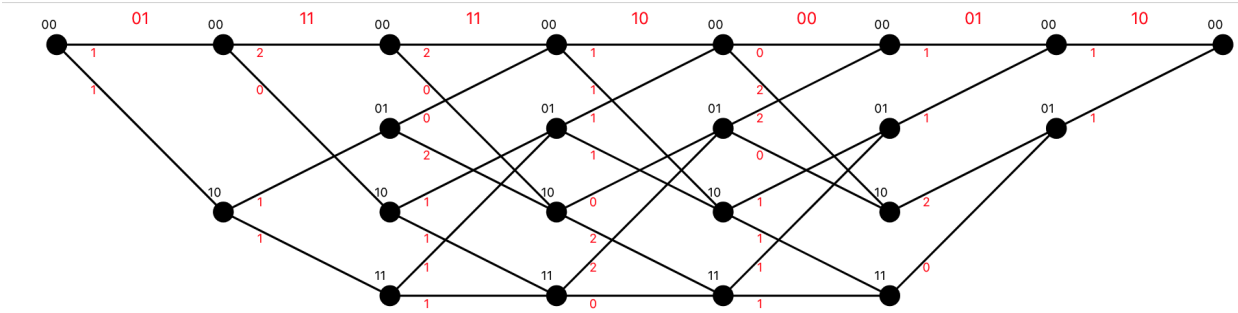
Each pair corresponds to one stage of the trellis. For every edge in the trellis, we compare the two bits associated with that edge to the corresponding received pair at that stage. The Hamming distance between these two pairs is called the **branch metric**, denoted by BM.

For example, if an edge outputs 11 while the received pair is 01, then

$$\text{BM} = d_H(11, 01) = 1.$$

Thus, every edge in the trellis is assigned a cost that measures how well that transition agrees with the received sequence. We first illustrate this for a single trellis stage.

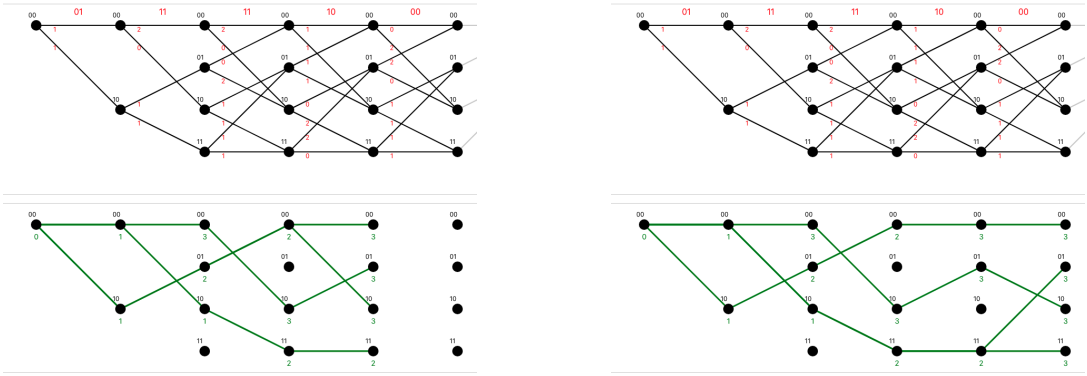
Repeating this process for every stage gives the full branch-metric trellis.



8.4.2 Path Metrics and Survivor Paths

Once the branch metrics have been assigned, we next compute the **path metrics**. We begin at state 00, since the encoder always starts in that state. At each time step j, we maintain four survivor paths: one ending at each of the four states 00, 01, 10, 11. For a state s at time j, the **survivor path** is the path that starts at 00, reaches state s at time j, and has the minimum total branch metric among all such paths.

The sum of branch metrics along this survivor path is called the **path metric**, denoted by PM.



Recursive Computation of Path Metrics We now explain how the path metrics are updated recursively. Suppose we have already computed the path metrics up to time j. To compute the path metric for a state s at time $j+1$, we consider its two predecessor states, say a and b. The two candidate path metrics are

$$\mathrm{PM}[a, j] + \mathrm{BM}[a \to s, j], \qquad \mathrm{PM}[b, j] + \mathrm{BM}[b \to s, j].$$

We retain the smaller of the two:

$$\mathrm{PM}[s, j+1] = \min\left\{\mathrm{PM}[a, j] + \mathrm{BM}[a \to s, j], \mathrm{PM}[b, j] + \mathrm{BM}[b \to s, j]\right\}.$$

The corresponding path is retained as the survivor path for state s. This recursive update rule is the dynamic programming step of the Viterbi algorithm. This is explained through an example in the above figure.

8.4.3 Final Decoding Rule

After processing all time steps, the decoded information bits are obtained from the survivor path ending at state 00 at the final time step. Recall that this final state is known because we append two zero bits during encoding.

8.4.4 End-to-End Viterbi Decoding Algorithm

We now summarize the complete decoding procedure from start to finish.

1. Divide the received bit sequence into consecutive pairs:

$$(r_1, r_2), (r_3, r_4), \ldots, (r_{2T-1}, r_{2T}).$$

2. For every edge in the trellis and every time step, compute the branch metric

$$\text{BM} = \text{Hamming distance between edge output and received pair.}$$

3. Initialize the path metrics at time 0:

$$\text{PM}[00, 0] = 0, \qquad \text{PM}[01, 0] = \text{PM}[10, 0] = \text{PM}[11, 0] = \infty.$$

4. For each time step $j = 0, 1, \ldots, T-1$:
   (a) For each state s at time $j+1$, consider its two predecessor states.
   (b) Compute the two candidate path metrics:

$$\text{PM}[a, j] + \text{BM}[a \to s, j], \qquad \text{PM}[b, j] + \text{BM}[b \to s, j].$$

   (c) Retain the smaller of the two as

$$\text{PM}[s, j+1].$$

   (d) Store the corresponding predecessor as the survivor path.

5. At the final time step, select the survivor path ending at state 00.

6. Trace back this survivor path from the final state to the initial state.

7. The decoded information bits are the input bits corresponding to the transitions along this path.

Visualization: An interactive visualization showing a step-by-step working of the Viterbi decoding can be found here.

8.4.5 Computational Complexity

At every time step, we update path metrics for only four states. Therefore, the total computational complexity grows linearly with the number of information bits, i.e., the complexity grows as $O(k)$. This linear-time complexity makes the Viterbi decoding much more practical than brute force decoding, where the decoding complexity grows exponentially.