CSCE 5760: Design For Fault Tolerance

HW #1. Take a look at the solutions
I will quickly summarize how to approach the problems

2.1. We are told that the system failed between 4 and 8 years. We need to find the probability that it failed before 5 years. That we need the conditional probability

\[ P(T<5|4<T<8) \]

failed between 4 and 5 years

\[
\frac{\text{Prob}(T<5\cap[4 \leq T \leq 8])}{\text{Prob}(4 \leq T \leq 8)} = \frac{F(5) - F(4)}{F(8) - F(4)} = \frac{(1-e^{-5\lambda}) - (1-e^{-4\lambda})}{(1-e^{-8\lambda}) - (1-e^{-5\lambda})} = 0.455
\]

Second problem is straightforward -> use Weibull distribution

Third problem involves converting the system into series-parallel combinations

---

CSCE 5760: Design For Fault Tolerance

Figure 2.2: A 5-module series-parallel system.

Let us consider the parallel set in the top middle (I will label the units as C and D)

\[ R_{CD} = 1 - (1-R_c)(1-R_d) \]

Now let us add the unit in front (say A)

\[ R_{ACD} = R_a R_{CD} = R_a(1 - (1-R_c)(1-R_d)) \]

Now we have two parallel paths (with the bottom left, say B)

\[ R_{BACD} = 1 - [(1-R_B)^a R_a(1 - (1-R_c)(1-R_d))] \]

Finally we add the unit on the right (say E)

\[ R_{ABECD} = R_b R_{ABCD} R_e \]

---
CSCE 5760: Design For Fault Tolerance

Another problem → related HW #2

Suppose that the reliability of a system consisting of 4 blocks, two of which are identical, is given by the following equation:

\[ R_{\text{system}} = R_1 R_2 R_3 + R_2^2 R_1 R_3 \]

draw the reliability block diagram representing the system.

\[ R_{\text{system}} = R_1 R_2 R_3 + R_2^2 R_1 R_3 = R_1 [R_2 R_3 + R_1 - R_1 R_2 R_3] \]

This means module M₁ is in series with some other network \[ [R_2 R_3 + R_1 - R_1 R_2 R_3] = 1 - [(1-R_2 R_3)(1-R_1)] \]

We have M₂ and M₃ in series and together in parallel with M₁.

CSCE 5760: Design For Fault Tolerance

Review:

Markov Chains and Markov Processes
Deriving state probabilities
Steady state
Absorbing states
Deriving differential equations

An example: Duplex with repair

\[ \frac{dP_1(t)}{dt} = -P_1(t) \sum_{j \neq 1} \lambda_{ij} + \sum_{j \neq 1} \lambda_{ji} P_j(t) \quad \text{Our general equation} \]

\[ \frac{dP_2(t)}{dt} = -2\lambda P_2(t) + \mu P_1(t) \]

\[ \frac{dP_4(t)}{dt} = 2\lambda P_2(t) + 2\mu P_0(t) - (\lambda + \mu) P_1(t) \]

\[ \frac{dP_0(t)}{dt} = \lambda P_1(t) - 2\mu P_0(t) \]

Initial condition

\[ P_2(0)=1, P_1(0)=P_0(0)=0 \]
CSCE 5760: Design For Fault Tolerance

Solving the differential equations we get

\[ P_2(t) = \mu^2/\lambda^2 (\lambda + \mu)^2 + 2\lambda\mu/\lambda^2 (\lambda + \mu)^2 e^{-2(\lambda + \mu)t} + \lambda^2/\lambda^2 (\lambda + \mu)^2 e^{-2(\lambda + \mu)t} \]

\[ P_1(t) = 2\lambda\mu/\lambda^2 (\lambda + \mu)^2 + 2\lambda(\lambda - \mu)/\lambda^2 (\lambda + \mu)^2 e^{-2(\lambda + \mu)t} - \lambda^2/\lambda^2 (\lambda + \mu)^2 e^{-2(\lambda + \mu)t} \]

\[ P_0(t) = 1 - P_2(t) - P_1(t) \]

Note that this Markov process is irreducible \(\rightarrow\) only recurrent states and no absorbing states.

If we have irreducible process, we can derive the steady state behavior.

That is when time \(t \rightarrow \infty\)

\[ dP_2(t)/dt = -2\lambda P_2(t) + \mu P_1(t) \]

\[ dP_1(t)/dt = 2\lambda P_2(t) + 2\mu P_0(t) - (\lambda + \mu) P_1(t) \]

\[ dP_0(t)/dt = 2P_1(t) - 2\mu P_0(t) \]

Setting \(dP(t)/dt = 0\) we get

\[-2\lambda P_1(t) + 2\mu P_0(t) = 0\]

\[2\lambda P_2(t) + 2\mu P_0(t) - (\lambda + \mu) P_1(t) = 0\]

\[\lambda P_1(t) - 2\mu P_2(t) = 0\]

\[P_0(t) + P_1(t) + P_2(t) = 1\]

And solving these linear equations we get

(we can drop \(t\) since we are looking at steady state)

\[ P_2 = \mu^2/\lambda^2 (\lambda + \mu)^2 \]

\[ P_1 = 2\lambda\mu/\lambda^2 (\lambda + \mu)^2 \]

\[ P_0 = \lambda^2/\lambda^2 (\lambda + \mu)^2 \]
CSCE 5760: Design For Fault Tolerance

So, the long term availability is given by

\[ 1 - P_0 = P_1 + P_2 = \left( \mu^2 + 2\lambda \mu \right) / (\lambda + \mu)^2 = 1 - \lambda^2 / (\lambda + \mu)^2 \]

Another way of looking at modeling continuous time Markov processes. Let us construct a transition matrix (instantaneous transition probability). Consider a system with two states: for example the following systems:

\[
M = \begin{bmatrix}
m_{11} & m_{21} \\
m_{12} & m_{22}
\end{bmatrix}
\]

Note how the rows and columns are numbered: columns should add to zero.

*If we have absorbing states, diagonal elements of those states will be zero.*

Using our example with 3 states we have:

\[
\frac{d}{dt} \begin{bmatrix}
P_1(t) \\
P_2(t) \\
P_3(t)
\end{bmatrix} = \begin{bmatrix}
-2\lambda & \mu & 0 \\
2\lambda & -\lambda - \mu & 2\mu \\
0 & \lambda & -2\mu
\end{bmatrix} \begin{bmatrix}
P_1(t) \\
P_2(t) \\
P_3(t)
\end{bmatrix}
\]

These are the same equations we had before.
CSCE 5760: Design For Fault Tolerance

What are watchdog processors? There are also watchdog timers.

Watchdog processors only check for correct execution flow. Check the control flow of a program.

Signatures with instructions executed by each basic block. So if some instructions are incorrectly sequenced, signature differs. Computed vs assigned.

Can we use similar idea for detecting security violations?

CSCE 5760: Design For Fault Tolerance

Malicious failures \( \rightarrow \) not benign.

The approach to solve such problems is called Byzantine algorithm -- related Byzantine generals stories.

Byz(N, m) -- N nodes with up to m failed nodes.

Step 1: Original source sends data to each of the N-1 receivers.
Step 2: If m=0, each of the N-1 receivers become sources. They distribute the values received in the previous step to other nodes.

In other words, each of the N-1 nodes apply Byz(N-1, m-1) algorithm.

-- step 2 is recursive

Step 3: At the end of communications, each node has a “vector” of values.
Each node looks for a majority value in this vector.
If no majority, need to a default value.

To correctly work, N >= 3m+1.
CSCE 5760: Design For Fault Tolerance

Some assumptions
- Non faulty unit is truthful about all its messages
- Faulty unit may send contradictory or even no messages
- Time out mechanism is available to detect “no message”
  - When no message, assume a default value

Chapter 3. Information redundancy → coding theory

Vector space – N dimensional space over a q elements
  - (that is each dimension can take q values) can have \( N^q \) vectors.
  - If binary space, \( q=2 \)

If we select a subspace – with fewer than the maximum vectors we defined a code

The idea is: the only legal vectors are those in the code space.
  - if you see a vector outside the code space – an error.

---

CSCE 5760: Design For Fault Tolerance

Simple parity (only for binary data)
- one extra bit \( p = 1 \) or \( c = d+1 \)
  - only half of all possible values are used

Distance (Hamming distance) between two codewords - the number of bit positions in which the two words differ

Minimum distance is the minimum of all distances between pairs of codewords
- For simple parity, minimum distance = 2

Distance can be defined for other bases (say decimal)

Simple introduction first
- To detect \( k \) bit errors, minimum distance must be \( \geq k+1 \)
- To detect 1 bit error minimum distance must be at least 2
  - Simple parity works

To correct (implies detect first) \( k \) errors, minimum distance must \( \geq 2k+1 \)
CSCE 5760: Design For Fault Tolerance

Consider a minimum distance 3 Hamming code with 4 bit binary numbers. We need to add 3 parity bits. So we have 7-bit code words for 4-bit data word.

\[
\begin{array}{ccccccc}
P_1 & P_2 & D_1 & P_3 & D_2 & D_3 & D_4 \\
\end{array}
\]

\[P_1\] is a parity on \(D_1, D_2\) and \(D_3\)
\[P_2\] is a parity on \(D_1, D_3\) and \(D_4\)
\[P_3\] is a parity on \(D_2, D_3\) and \(D_4\)

Consider data value of 0010
\[P_1 = 0; P_2 = 1\text{ and } P_3 = 1\]
Codeword = 0101010

How do we detect and correct errors

Compute parities from received data – if all parities check, no error
If parity \(P_i\) does not match correct value, we assign a \(S_i = 1\)
So, we have 3 bit Syndrome \(S_1S_2S_3\) which ranges between 000 and 111
This number locates the error bit

CSCE5760  September 24, 2019  13

---

CSCE 5760: Design For Fault Tolerance

In general if we have \(N = 2^n\) data bits, we need \(n+1\) parity bits (distance 3 code)
For 32 bit data we need 6 parity bits
Parity bits are located at bit positions corresponding powers of 2

\[P_i\] is at bit position \(2^i\)
\[P_i\] at bit 1; \(P_2\) at bit 2, \(P_3\) at bit 4; \(P_4\) at bit 8; \(P_5\) at bit 16; \(P_6\) at bit 32

\[
\begin{array}{cccccccc}
P_1 & P_2 & D_1 & P_3 & D_2 & D_3 & D_4 & P_4 \\
\end{array}
\]

Parity \(P_i\) is a parity on “alternating 2 bits, starting \(P_i\)

\[P_1\] is parity on \(P_1, D_1, D_2,\ldots\)
\[P_2\] is parity on \(P_2, D_1, D_3, D_4, D_6, D_7,\ldots\)
\[P_3\] is parity on \(P_3, D_2, D_3, D_4, D_5, D_6, D_8, D_{10}, D_{11},\ldots\)
\[P_4\] is parity on \(P_4, D_5, D_6, D_7, D_8, D_{10}, D_{11},\ldots\)

CSCE5760  September 24, 2019  14
CSCE 5760: Design For Fault Tolerance

Separable codes
- data and parity bits are separate
- Hamming code is a separable code

Non separable codes
- data and parity cannot be separate
- more complex to decode

Groups, Rings and Fields

A group \([S, +]\) is a non-empty set with a binary operation (operation on two values):
- closure \((a+b)\) is also a member of the group
- associative \((a+b)+c = a+(b+c)\)
- an identity element exist for the operation \((a+0 = a)\)
- each element has an inverse under the operation \((a+(-a)) = 0\)

\textbf{commutative group if the operation is commutative} \((a+b=b+a)\)

Examples: Set of all integers with addition (not just positive integers)
- Polynomials in x. Addition and multiplication of polynomials.
- Modulo addition and multiplication.

CSCE 5760: Design For Fault Tolerance

Ring, \([R, +, \ast]\) is a ring on a set of elements \(R\) defined with two operations, + and \(\ast\).
- \([R, +]\) must be a commutative group.

and the \(\ast\) operation must satisfy: closure, associative and distributive operation (over +)
- Distributive: \(a^\ast(b+c) = a^\ast b + a^\ast c\) no need for an inverse

Examples: Set of Integers (both positive and negative) with addition and multiplication is a Ring.
- Polynomials over real (or integers) is a ring under addition and multiplication.

Another example:
Consider the set of \(n\times n\) matrices over integers.
Define matrix addition and Matrix multiplication
- not regular matrix multiplication but multiplication of respective elements

We have ring here.
Yet more complex structure is called a **field**. \([F, +, \ast]\) is a field if \([F, +, \ast]\) is a ring

that is, the following hold

- **Commutative** \(a+b = b+a\) and \(a\ast b = b\ast a\)
- **Associative** \((a+b)\ast c = a + (b+c)\) and \((a\ast b)\ast c = a\ast (b\ast c)\)
- **Distributive** \(a\ast(b+c) = a\ast b + a\ast c\)

and

- **Field has both additive and multiplicative identity elements**
  \(a+0 = 0; \quad a\ast 1 = a\) (0 and 1 are identify elements)
- **Fields have additive and multiplicative inverses (except for 0)**
  \(a+b = 0\) then \(b\) is the inverse of \(a\) under addition
  \(a\ast b = 1\) then \(b\) is the inverse of \(a\) under multiplication

**Examples:** The set of real numbers under **addition** and **multiplication** is a field

(note set of integers is not – why not?)

Consider **Modulo addition** and division. \([\mathbb{Z}_5, +, \ast]\). Is this a field?

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\ast</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Important:** In a field, \(a\ast b = 0\) if and only if either \(a\) or \(b\) is 0 (0 is the additive identity).

Consider the following structure. \([(0,1,2,3), +, \ast]\). That is, modulo 4 arithmetic.

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\ast</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
What is the inverse of 2?
2 has no inverse. So this is not a field.

Why is \( \mathbb{Z}_5 \) a field but not \( \mathbb{Z}_4 \)?
\( \mathbb{Z}_p \) is a field if \( p \) is a prime number.

Consider the following however. Instead of \( \{0, 1, 2, 3\} \) let us use the following numbers. \( \{0, 1, a, b\} \). We will perform modulo 2 arithmetic.

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>*</th>
<th>0</th>
<th>1</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Now we have a field.
This because, we are not looking at \( \mathbb{Z}_n \), we are looking at \( \mathbb{Z}_x \) where \( x = 2^2 \)
Or in other words, a filed over \( x = p \) where \( p \) is a prime number.

We need to be careful in defining these tables for + and \( \ast \) to make sure we have a field
Such fields are called Galois fields.

We will write them as \( GF_q \) or \( GF(q) \) where \( q = p^h \) and \( p \) is a prime number

Another interesting factor. For any prime power field, there is a non-zero element say \( g \) that generates all the other members: that is, all the other number can be written as \( g^i \).
Such an element is called the primitive element (or generator element).
In the above example (with 4 elements 0, 1, a, b), a can be used as the primitive number since
\[
\begin{align*}
a^0 &= 0, & a^1 &= a, & a^2 &= a^1 a = b, & a^3 &= b^2 a = 1. 
\end{align*}
\]

More complex structures. **Vector Spaces**. We have to first start with a field, that is we will start with say \([F, +, \ast]\) that gives us a field (the set \( F \) contains values taken by field elements). We will define vector of dimension \( n \) as a \( n \) tuple.
\[
v = (v_1, v_2, \ldots, v_n) \rightarrow \text{all } v_i \text{ elements belong to } F
\]

We can now define vector addition as a binary operation.
\[
u + v = (u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n)
\]

Is this operation closed?
Since the tuple elements are from the field and addition is closed on these elements, vector addition is also closed.
Likewise this operation is associative.

The vector \( 0 = (0, 0, ..., 0) \) the zero vector is the identity for the vector addition operation.

What about inverse of a vector?

\[ V^{-1} = (v_1^{-1}, v_2^{-1}, ..., v_n^{-1}) \]  \( \rightarrow \) this vector is in the same field since \( v_i^{-1} \) belongs to F

We can also define a scalar multiplication on vectors.

\[ a^*V = (a^*v_1, a^*v_2, ..., a^*v_n) \]

Since \( a^*v_i \) is defined on the underlying field elements, this scalar multiplication is closed, and associative.

1 (the multiplication identity of the underlying field) is the identity for scalar multiplication

Scalar multiplication is associative: \( (c^*d)^*V = c^*(d^*V) \)

Distributive Laws

\[ c^*(U+V) = c^*U + c^*V \]
\[ (c+d)^*V = (c^*V + d^*V) \]

For scalar multiplication we cannot define inverses, since the identity is not a vector.

We can sometimes define vector multiplication (this is different from dot product)

\[ U^*V = (u_1^*v_1, u_2^*v_2, ..., u_n^*v_n) \]

Now can define an identity vector \((1, 1, ..., 1)\)

This vector multiplication is closed, associative

\[ U^*(c^*V + d^*W) = c^*(U^*V) + d^*(U^*W) \]

Then we have a linear associative algebra.
Linear combination of vectors.

\[ V = a^*A + b^*B + \ldots \]
where A and B are vectors themselves.

In other words, we can express one vector as a combination of other vectors.

If we start with say n vectors \( V_1, V_2, \ldots, V_n \),
we can define a vector space by taking all possible linear combination of these vectors.

The set of vectors \( V_1, V_2, \ldots, V_n \) are known as linearly independent if we cannot express any one of the vectors as a linear combination of the others vectors. \( \Rightarrow \) basis vectors

We can talk about Subgroup, we can talk of subspace, subfields etc.

Subgroup. \([H, +]\) is a subgroup of \([G, +]\) if H is a subset of G and H is a group under +.

That is, + must be closed within H, must be associative, must have an identity and must define inverses for the elements of H. But, we need only to check for closure and inverses (the other properties follow).

Examples. Consider the set of \( \{0,1,a,b\} \) with the addition defined – here we have \( q=2^2 \)

\[ \{0,1\} \text{ forms a subgroup under the same operations. But } \{a,b\} \text{ does not.} \]

--- does not contain the additive identify

In general for subgroup \( H = \{h_0, h_1, \ldots, h_{m-1}\} \), m is the cardinality of the set (also known as the rank or order of the group)

The identity element of \( G \) is also the identity element of \( H \).

What is the cardinality of group \( G \)?
Must be an integer multiple of m. Why?
CSCE 5760: Design For Fault Tolerance

Consider writing as follows (each $g_i$ is a member of the group but not in $H$)

\[
\begin{array}{cccc}
h_0 & h_1 & h_2 & \ldots & h_m-1 \\
g_1+h_0 & g_1+h_1 & g_1+h_2 & \ldots & g_1+h_{m-1} \\
g_2+h_0 & g_2+h_1 & g_2+h_2 & \ldots & g_2+h_{m-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
g_k+h_0 & g_k+h_1 & g_k+h_2 & \ldots & g_k+h_{m-1} \\
\end{array}
\]

Now we have rows – all of which are members of the group – but not of the subgroup $H$ (except first row).

So, the cardinality of $G$ is an integral multiple of the cardinality of $H$.

Each row of elements $\{g_i+h_i \mid h_i \in H\}$ is called a co-set of $H$,
and $g_i$ is called the co-set leader.

Co-sets are useful in detecting and correcting errors.

CSCE 5760: Design For Fault Tolerance

How do we know if two elements $g$ and $g'$ are in the same co-set?
If they are, then we can use the co-set leader say $g_j$ and express $g$ and $g'$ as
\[
g = g_j+h_i \quad \text{and} \quad g' = g_j+h_k
\]

Consider the element $g^{-1}+g' = (g_j+h_i)^{-1}+(g_j+h_k) = (h_i^{-1}+g_j^{-1})+(g_i+h_a) = (h_i^{-1}+h_a)$

Since $h_i$ and $h_a$ are element of $H$, $(h_i^{-1}+h_a)$ must also be an element of $H$.

In other words, if $g^{-1}+g'$ belongs to $H$ then $g$ and $g'$ belong to the same co-set.

Normal Subgroup. $H$ is a normal subgroup of $G$ if and only if for every element $g$ of $G$ and any element of $h$, $g^{-1}+h+g$ is in $H$.

Vector Subspaces. Any linear combination of vectors, say $V_1, V_2, \ldots, V_n$ forms a subspace.
If we can obtain the same vector space with $k$ linearly independent vectors $U_1, U_2, \ldots, U_k$ then we call these $k$ vectors as the basis and the vector subspace describes a $k$ dimensional space.

We can represent any vector in the $k$-dimensional space as a linear combination of the basis vectors.

$$V = (a_1*U_1 + a_2*U_2 + \ldots + a_k*U_k)$$

We can also represent this linear combination as

$$(a_1, a_2, \ldots, a_n) [U]$$

where $[U]$ is a matrix written as

$$
\begin{bmatrix}
U_1 \\
U_2 \\
U_3 \\
\vdots \\
U_k
\end{bmatrix}
$$

We should know that two Matrices $M$ and $N$ are equivalent if one can be obtained from the other by using 1) row transposition, 2) row arithmetic, 3) column transposition and 4) column arithmetic.

For example consider the following matrices in binary field

$$
\begin{bmatrix}
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 0
\end{bmatrix}
$$

These matrices are equivalent (add row 1 to row 2; add row 1 and row 2 to row 3).

For any $k$-dimensional vector space, we can use the following basis

$$(1,0,0,\ldots,0); (0, 1,\ldots,0); (0,0,\ldots,1).$$

In matrix representation this looks like the Identity matrix.

However, we can also represent these vectors as matrices, Consider the matrices above

We have 3 independent rows and thus 3 independent vectors

We can generate vectors in 3 dimensional space (subspace of 4 dimensions)
A linear code is nothing but a subspace. Every element (or codeword) of a linear code can be expressed as a linear combination of the basis vectors or basis codewords.

The basis codewords can be written as a matrix.

Consider the following Hamming code.

\[
\begin{array}{cccc}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\end{array}
\]

Either of these matrices form the basis for the 4-dimensional vector subspace in a 7-dimensional space. And give the code we are looking for.

The left matrix looks like \{I | \mathbf{P}\} where I is the identify matrix.

In order to construct a linear code, we need to consider a vector space (actually a subspace) over a finite Galois Field (filed over set containing a prime number of elements or power of a prime set)

If we are looking at a k dimensional subspace of a n-dimensional vectors, we will call the code \([n, k]\) code over \(\text{GF}_q\).

Sometimes we would also describe the "minimum distance" \(d\); \([n, k, d]\) code.

Note since the linear code \(\mathbf{C}\) is a subspace, for any two vector \(\mathbf{U}\) and \(\mathbf{V}\) in \(\mathbf{C}\), \(\mathbf{U}+\mathbf{V}\) is also in \(\mathbf{C}\) (since the vector addition must be closed).

And for any scalar \(a\) in \(\text{GF}_q\), \(a*\mathbf{V}\) is also in \(\mathbf{C}\) (scalar multiplication is closed).

The zero vector (0,0,...,0) always belongs to any linear code -- the identity over vector addition.

Thus the codewords (elements of \(\mathbf{C}\)) are n-dimensional vectors forming a k dimensional vector subspace.

In terms of codes, \(n-k\) is the number of parity bits or redundancy, \(k\) is the number of data bits.

\(n-k\) is the redundancy.
Weight of a vector or a codeword is the number of non-zero elements of the vector (represented as a tuple).

For example, consider a 4 dimensional vector over GF3 –
(1, 0, 2, 0) weight is 2  (2, 2, 2, 1) weight is 4.

Distance between two vectors: is the weight of the difference vector.

\[(1,0,2,0) - (2,2,2,1) = (1,0,2,0) + (-2, -2, -2, -1) = (1,1,1,2) = (2,1,0,2)\]

Difference = weight of \((2,1,0,2) = 3.\)

Note. Weight of a binary vector is the number of 1’s in the vector
Distance between 2 binary vectors is the number places they differ

\[
\begin{align*}
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & \text{weight = 3} \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & \text{weight = 4} \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & \text{weight = 3} \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & \text{weight = 3}
\end{align*}
\]

Distance [(1000101), (0100111)] = weight (1100010) = 3

Minimum distance of a vector space (or a code — vector subspace)
The minimum of all distances between any pair of vectors or codewords.

Note, the difference between any two codewords is a codeword since vector addition/subtraction is closed.

So, minimum distance of a code is the minimum weight of a codeword in the code.

How to generate codewords. A code is a vector space. We can think of basis vectors which can be written as generator matrix using the basis vectors.

For a \([n,k]\) code -- k-dimensional subspace of n-dimensional space, we need \(k\) basis vectors; each vector is a n tuple. That is we have \(k \times n\) matrix.

Since we are dealing with k-dimensional subspace, we can have only \(q^k\) codewords (vectors) in our code.

Since we can manipulate matrices and obtain a new equivalent matrix, it will be useful to convert a generator matrix of a code to look like \([I_k \; A_k \; x \; n-k]\) matrix. (\(I_k\) is identity matrix)
Consider the Hamming code example

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\quad \begin{array}{cccc}
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

This is what we want to generate

We can use either of these generator matrices.
The one on the right is how I have shown previously when we generated parity bits for Hamming code.

This is a [7, 4] code. We can have only \(2^4 = 16\) codewords.
We can encode 4 bit numbers since we have 16 different possible numbers here.
We can encode 4-bit numbers to get 7-bit codewords --- 3 parity bits.
To do this all we have to do is multiply the 4 bit number by the generator matrix.

The same applies for any [n,k] code over any GFq field.

How to decode?

Note that Vector spaces forms a group under vector addition. Since a Code is a subspace, it forms a subgroup under vector addition.

It means we can construct co-sets using the elements of the code and the vectors which are not part of the code.
These vectors which are not part of the code indicate errors.

The co-set leader (the vector with the smallest weight in a co-set) indicates the error or syndrome.

We could create an array.

For example with 7-bit binary numbers and 16 code words, we would have 7 co-sets in addition to the code -- note the 7 co-sets indicate the 7 possible error states we discussed.

So, we can construct an array 8 x 16 (7 co-sets and the codewords)

We can then locate where the received (code with error) falls and appropriately decode it.
CSCE 5760: Design For Fault Tolerance

In general, we should compute for each syndrome, the coset leader.

How to find co-sets?

We can write the co-set as

\[
\begin{array}{cccc}
  h_0 & h_1 & h_2 & \ldots & h_m \\
  g_i + h_0 & g_i + h_1 & g_i + h_2 & \ldots & g_i + h_m \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  g_k + h_0 & g_k + h_1 & g_k + h_2 & \ldots & g_k + h_m \\
\end{array}
\]

That is, the first row is our code words

For every \( g \) not in the code, we create a new row and \( g \) is the co-set leader

For \( g \), compute syndrome.

For any \( y \), compute syndrome – then use the co-set array to locate the row corresponding to the syndrome – find \( y \) in that row.

The correct code is found in the first row for that column

CSCE5760 September 24, 2019

CSCE 5760: Design For Fault Tolerance

However, this is too slow or takes a lot of memory. Alternative?

**Dual Code space.** For each linear code, we can construct another code that is an orthogonal space.

That is, if we have a code space \( C \), we can build another code space \( D \) which is perpendicular to \( C \).

Or, if a code (vector) is in \( C \), it is not in \( D \), and vice versa. But \( D \) is a linear code by itself.

If a code \( C \) is \([n,k]\) subspace, its orthogonal dual will be \([n,n-k]\) subspace.

How do we find this orthogonal space?

If \( G \) is the generator matrix for \( C \), the generator matrix for \( D \) (say \( H \)) satisfies

\[ G \times H^T = 0; \quad \text{if } G = [I \mid A] \text{ then } H = [A^T \mid I]. \]

We call \( H \) the **parity check matrix**.
CSCE 5760: Design For Fault Tolerance

Assume we received a data item y (a bit value). If this is correct, then it must belong C and if not in D.

If we multiply y by $H^T$, we should get a zero vector if y is correct. If we get a non-zero, then there is an error and $y \cdot H^T$ is called a syndrome.

For each syndrome, we can identify a co-set in which y belongs (hence we can decode).

Consider the [7,4,3] (dimension of vector space, dimension of code space and minimum distance) Hamming code – use the following generator matrix:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

What is the parity check Matrix H?

\[
G = [I \mid A] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}
\]

\[
H = [A^T \mid I] = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}
\]
Consider our Hamming code \([7,4,3]\) – 3 parity bits
What are the possible syndromes?

<table>
<thead>
<tr>
<th>Syndrome</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 000</td>
</tr>
<tr>
<td>000 001</td>
</tr>
<tr>
<td>000 010</td>
</tr>
<tr>
<td>000 100</td>
</tr>
<tr>
<td>001 000</td>
</tr>
<tr>
<td>001 010</td>
</tr>
<tr>
<td>001 100</td>
</tr>
<tr>
<td>010 000</td>
</tr>
<tr>
<td>010 010</td>
</tr>
<tr>
<td>010 100</td>
</tr>
<tr>
<td>011 000</td>
</tr>
<tr>
<td>011 010</td>
</tr>
<tr>
<td>011 100</td>
</tr>
</tbody>
</table>

Note the syndrome indicates the error state as represented by the parities
And the co-set leader indicates which bit is in error (or just subtract this co-set from the received value)

Consider the correct codeword \([1010101]\) but we received \([1010111]\)
\(y^*H^T = [001]\) gives us the syndrome

Remember co-sets and co-set leaders?
We can think of creating a matrix as we did before.

Now we can use this matrix to find the co-set leader based on the syndrome.
Then we can subtract the co-set leader from received value to get original codeword

Let us see how we can construct H matrix for any number of data bits
Note that if we can construct the H matrix, we can get G matrix for any code

Suppose you want to create a code for \(d\) data bits
note that we need \(r\) parity bits where \(2^r \geq d+r+1 \rightarrow\) number of states
and \(n = d+r\)

Here we are looking at distance 3 (or only one bit is in error hence the number of states is \(d+r+1\))
CSCE 5760: Design For Fault Tolerance

H will have r rows and n columns.
We can start with H by using all non zero values for columns
Then rearrange H to look like \([A^T\mid I]\) and then obtain G

Consider \(r = 3\) (\(n=7\) and \(d=4\))
Let us start H as

\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{bmatrix}
\]

Each column is a non-zero value with 3 bits

We can rewrite this into

\[
\begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\]

Then G =

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 2 \\
\end{bmatrix}
\]

CSCE 5760: Design For Fault Tolerance

For any field over \(q\), we need to create H using non zero values possible as columns. For example consider the filed over \((0, 1, 2)\) or \(q=3\)
Let us use \(2\) parity digits \((r=2)\)

\[
3^2 \geq d + r + 1
\]

So \(2^3 \geq d + r + 1\)

Now we can have \(6\) data digits
One example H matrix can be \((2^2\times 8)\)

\[
\begin{bmatrix}
0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\
1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
\end{bmatrix}
\]

We can rewrite the H to be in the standard form \([A^T\mid I]\)
and then get G matrix