Comunicaciones Inalámbricas

Notas del Curso

Facultad de Ingeniería
Universidad de la República

Pablo Belzarena y Federico Larroca

3 de octubre de 2017

Índice general

Capítulo 1

Codificación

1.1. Introducción

El tema de este capítulo, corrección de errores, es un tema de muchos años de desarrollo en el área de las comunicaciones y la electrónica. Es por tanto, un tema que se encuentra muy bien abordado en diferentes textos y cursos. En estas notas se pretende dar una introducción al tema haciendo énfasis en aquellos aspectos que interesan para el curso de comunicaciones inalámbricas. En diversos libros de comunicaciones inalámbricas (por ejemplo [3], [4]) este tema está tratado y se han tomado de ellos la forma de abordar algunas de las secciones de este capítulo. También en el curso del MIT [6] hay una buena introducción a la corrección de errores que ha sido de utilidad también para ver la forma de explicar algunos temas. Un estudio más profundo de este tema se puede encontrar el libro de Lin y Costello [5] y que ha sido referencia para estas notas en varias secciones. Este tema tiene relación con el fundamento teórico en la Teoría de la Información. En este curso se estudiarán solamente algunas cosas mínimas de esta teoría buscando hacer menos árida la presentación del tema. El lector interesado puede consultar diferentes libros sobre teoría de la información (por ejemplo [1] o una introducción más básica en [2]).

La gestión de errores es un tema muy relevante en cualquier sistema de comunicación pero particularmente en los sistemas de comunicación inalámbricos donde la tasa de errores del medio físico es importante. Por un lado, se utilizan los códigos de detección de errores cuyo objetivo es detectar (aunque no sea posible corregir) los errores generados en la comunicación. Por otro lado, se encuentran los códigos de corrección de errores que son diseñados no solo para detectar sino también para corregir errores.

Cuando se utilizan códigos para detectar errores, estos habitualmente se emplean junto con un mecanismo de requerimiento de repetición automática (automatic repeat request, ARQ). Este mecanismo puede funcionar de diferentes maneras, pero habitualmente, el receptor envía un ACK si una trama de datos llegó sin errores (o más precisamente sin que se hayan detectado errores) y cuando el transmisor no recibe el ACK retransmite la trama. Este mecanismo si bien permite solucionar el problema generado por la introducción de errores en una trama, agrega retardos que pueden ser importantes si el medio físico introduce muchos errores y por tanto si bien se logra un sistema de comunicaciones sin errores o con baja tasa de errores, los retardos

introducidos pueden no ser aceptables. Por este motivo, particularmente en las redes inalámbricas es muy importante buscar mecanismos que permitan corregir errores sin necesidad de realizar retransmisiones.

Si bien este capítulo se concentrará en los códigos de corrección de errores, es importante resaltar que en general los códigos de detección de errores y de corrección de errores se utilizan conjuntamente aunque muchas veces a diferentes niveles. Por ejemplo se utilizan códigos de corrección de errores en ciertos bloques y luego la trama globalmente tiene un CRC para detectar por si algún error no pudo ser corregido.

1.2. Introducción a los códigos de corrección de errores

Existen muchos tipos de códigos de corrección de errores. Este capítulo se centrará particularmente en dos clases de las muchas que existen. La primera clase que se analizará es la de los códigos lineales de bloques, que son una subclase de los códigos algebraicos. Estos códigos han sido ampliamente utilizados sobre todo para corregir errores en la escritura/lectura de dispositivos electrónicos (memorias, discos, etc.). Como se verá más adelante, hoy en día en sistemas de comunicaciones inalámbricos no son los mas utilizados. Sin embargo, estos códigos y algunas subclases como los códigos cíclicos siguen siendo utilizados y además didácticamente ayudan a comprender el funcionamiento de otros tipos de códigos por lo que se analizará su operación en primer término. En segundo lugar se analizarán los códigos convolucionales. Esta clase de códigos y derivados de ella son muy utilizados en comunicaciones inalámbricas. Por ejemplo GSM, 802.11, ISDBT, utilizan códigos convolucionales para corrección de errores en las comunicaciones.

Antes de analizar las clases de códigos antes mencionados, se presentarán algunas propiedades y características de la codificación en un canal de comunicaciones.

1.3. Modelo de canal, tasa de información, probabilidad de decodificación errónea

1.3.1. Canal BSC

Se denominará p_e a la probabilidad de error de un solo bit. Se asume que los bits son corrompidos de manera independiente y todos con la misma probabilidad p_e. Esto es lo que se denomina, canal binario simétrico (BSC) y con este modelo se trabajará en lo que resta de este capítulo. Para una trama de S bits en un BSC, la probabilidad de que sea recibida correctamente, será entonces: $(1 - p_e)^S$. En general se asume también que $p_e < 0{,}5$, esto además de corresponder a un canal 'razonable' tiene una justificación que se verá más adelante.

1.3.2. Una primera aproximación: códigos de repetición

¿Cuál es la primera idea que se le ocurre al lector para aumentar la confiabilidad de un canal con errores? Probablemente la primera idea sea repetir varias veces cada

bit a enviar. Por ejemplo si se quiere enviar un 0 se envía 000 y si se quiere enviar un 1 envío 111. En el receptor si hay errores en el canal se podrá recibir cualquier combinación de 3 bits.

La siguiente pregunta que se plantea es: ¿ Cómo se decodifica? es decir, si se recibe por ejemplo 010 qué se decidiría que se envió: ¿0 o 1?. La respuesta intuitiva es que se decodificaría como un 0 porque lo más probable es que exista un solo error y en ese caso lo que se envió debe haber sido 000. Esta respuesta intuitiva, se puede formalizar y se formalizará más adelante. Se analizará que la regla de asignación de máxima verosimilitud para decodificar, es aquella que asigna la palabra de código que está a menor distancia de la palabra recibida. Se probará para un canal BSC con probabilidad de error de bit menor a 0,5, con lo cuál también se justificará nuestra hipótesis inicial de que se trabajará en un BSC con $p_e < 0{,}5$.

Se supone entonces un BSC con $p_e < 0{,}5$ y donde se utiliza el código de repetición por tres y con la regla de decodificación comentada antes. ¿Cuál es la probabilidad que no existan errores, cuál que se cometa uno y solo uno, cuál que se comentan dos y solo dos errores y cuál que se cometan tres errores? El lector puede verificar que la probabilidad de que no se cometa ningún error es $(1-p_e)^3$. La probabilidad de que exista un solo error es $\binom{3}{1}p_e(1-p_e)^2$ (donde $\binom{3}{1}$ indica combinaciones de tres tomadas de a uno). La probabilidad de que existan dos y solo dos errores es $\binom{3}{2}p_e^2(1-p_e)$ y de que existan 3 errores es p_e^3.

Por lo tanto, la probabilidad de que una palabra en el receptor se interprete erróneamente es igual a la probabilidad de que existan 2 o 3 errores, esto es: $3p_e^2(1-p_e)+p_e^3$. Para p_e pequeño esto es aproximadamente igual a $3p_e^2$ y si por ejemplo $p_e = 0{,}01$, la probabilidad de que el mensaje se interprete mal es: $3\ 10^{-4}$. Es decir que repitiendo 3 veces cada bit, se reduce en dos órdenes de magnitud la probabilidad de error. Sin embargo, esto se logra al costo de reducir a 1/3 la tasa de información que se envía, ya que por cada 3 bits se envía un solo bit de información.

Si el lector realiza este ejercicio para un código de repetición de 5 o 7 bits, verá que la probabilidad de error bajará sustancialmente pero también a costa de bajar la tasa de información a 1/5 y 1/7 respectivamente. Por lo tanto, generalizando el razonamiento anterior, con un código de repetición es posible hacer que la probabilidad de error se acerque a cero tanto como se quiera pero a costa de acercar a cero también la tasa de información enviada.

La pregunta que se presenta inmediatamente es si existirá una forma de hacer asintóticamente nula la probabilidad de mensaje erróneo, pero sin tener que hacer asintóticamente nula la tasa de información enviada.

La respuesta a la pregunta anterior la dio Shannon en la década de 1950, y la respuesta es que efectivamente es posible hacer la probabilidad de mensaje erróneo asintóticamente cero sin que necesariamente la tasa de información tienda a cero.

En este capítulo no se verá la demostración del segundo teorema de Shannon, el cual puede verse cualquier libro sobre Teoría de la Información (ver por ej. [1]).

El Teorema dice que para una canal discreto sin memoria (esto es más general que el canal BSC que se está considerando), todas las tasas por debajo de la capacidad del canal C son alcanzables. Esto es que para toda tasa de información $R < C$, existe algún código cuya probabilidad de error tiende a cero con n (el largo de las palabras del código).

La capacidad del canal es un concepto de Teoría de la Información y solo se

recuerda al lector su definición: es la máximo de la información mutua entre transmisor y receptor. Por ejemplo, para un canal binario simétrico la capacidad del canal se calcula a partir de la definición de información mutua y se obtiene:

$C = 1 + p_e \log(p_e) + (1 - p_e) \log(1 - p_e)$

Notar que cuando $p_e = 0$ la capacidad es 1 y cuando $p_e = 0{,}5$, la capacidad del canal es nula, ya que cuando llega un bit no tengo ninguna información sobre el bit que fue enviado.

1.3.3. Distancia de Hamming, capacidad de detectar y corregir errores de un código

Una forma de pensar sobre cómo elegir las palabras de código es asumir que se utilizarán palabras de cierto largo N y que se tienen que elegir M de estas palabras para transmitir los $\log(M)$ bits de información que es necesario enviar.

Ejercicio 1.1. *Por ejemplo en el caso del código con repetición $N = 3$, $M = 2$. Cuando $M = 2$ se eligieron como palabras a utilizar de las 8 posibles $000, 111$. Si se asume que se produce un solo error ¿es posible corregir el error? ¿de qué manera? y si se pueden producir dos errores ¿es posible corregir los errores? ¿es posible detectar que hubieron errores? Explique porqué. Explique porqué es conveniente tomar esas dos palabras y no por ejemplo $000, 110$*

Ahora bien, si la cantidad de palabras que se quieren enviar fuera $M = 4$, es necesario elegir 4 palabras del conjunto: $000, 001, 010, 011, 100, 101, 110, 111$. ¿cuáles elegiría? ¿es posible corregir algún error? ¿es posible detectar que hubo un error? Expliqué porqué.

Definición 1.1. Distancia de Hamming para dos secuencias de N bits w_1 y w_2. Es el número de bits en que difieren ambas palabras.

Con esta definición se puede probar fácilmente que es una métrica en el espacio de las palabras de N bits. Esto quiere decir que:

1. $H(w_1, w_2) \geq 0$, en particular $0 \leq H(w_1, w_2) \leq N$
2. $H(w_1, w_2) = 0$ si y solo sí $w_1 = w_2$
3. $H(w_1, w_2) = H(w_2, w_1)$
4. $H(w_1, w_3) \leq H(w_1, w_2) + H(w_2, w_3)$, desigualdad triangular.

Ejemplo 1.1. $w_1 = 000000$ *y* $w_2 = 101010$ $H(w_1, w_2) = 3$.

Definición 1.2. Distancia mínima de Hamming de un código. Es la menor distancia de Hamming entre dos palabras cualesquiera del código.

Ahora bien, si se tienen dos palabras tales que $H(w_1, w_2) = 1$, si se produce un error en el bit en que difieren no será posible saber si se cometió un error. Si la distancia mínima de un código es 2, esto quiere decir $H(w_i, w_j) \geq 2$ para todo w_i, w_j del conjunto S de mensajes posibles del código. En este caso, si se transmite w_i y hay un error la palabra recibida r no puede pertenecer a S ya que de ser así la distancia mínima del código sería 1 y no 2. Por lo tanto, en este caso se puede detectar un

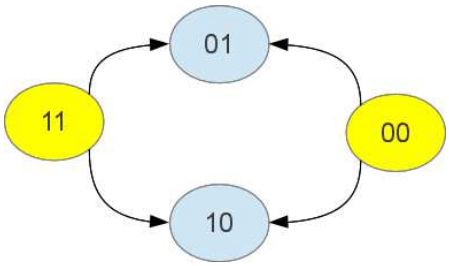

Figura 1.1: Diagrama ejemplo 1.2 .

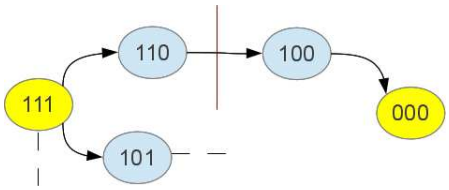

Figura 1.2: Esquema de código con distancia mínima de Hamming 3

error. En cambio con el mismo razonamiento, se puede ver que no todo patrón de dos errores puede ser detectado.

El razonamiento anterior se puede generalizar fácilmente en el siguiente teorema:

Teorema 1.1. *Un código con distancia mínima de Hamming D puede detectar $D-1$ o menos errores. Además, existe al menos un patrón con D errores que no puede ser detectado.*

Ejemplo 1.2. *Se considera el caso $N = 2$ y $M = 2$. Se eligen como palabras del código $(00, 11)$. En la figura 1.1 se muestra una representación del código donde cada transición representa un error en un bit, es decir que dos palabras unidas por una transición tienen una distancia de Hamming igual a 1. Como se puede observar en este caso el código tiene distancia mínima 2 y por tanto se puede detectar 1 error. Sin embargo, no es posible corregir porque las palabras erróneas se encuentran a la misma distancia de Hamming de las dos palabras de código.*

Se considera entonces el código de repetición ya visto y del que se muestra un diagrama parcial de transiciones en la figura 1.2. En este caso si se producen uno o dos errores la palabra recibida no pertenecerá a S (el conjunto de las palabras del código) porque la distancia mínima de este código es 3. Además, si se produce un error es posible saber qué palabra lo mandó ya que cualquier otra palabra de código estará a distancia mayor a uno. En cambio ¿es posible corregir si se producen dos errores?

Teorema 1.2. *Un código con distancia de Hamming mínima D puede corregir cualquier patrón de errores de $\frac{D-1}{2}$ o menos errores.*

Demostración. La idea para probar este Teorema es considerar el conjunto G_{w_i} que resulta de cambiar $\frac{D-1}{2}$ o menos bits a una palabra de código w_i. Si la distancia

mínima de Hamming es D, entonces $G_{w_i} \cap G_{w_j} = \emptyset \; \forall w_j$ porque si no fuera así por la desigualdad triangular habría dos palabras de código a distancia menor que D. □

Para terminar esta sección se presentará un teorema que se ha utilizado anteriormente de manera intuitiva.

Teorema 1.3. *En un canal BSC con probabilidad de error menor a 0.5, la estrategia de decodificación de máxima verosimilitud es mapear la palabra recibida a la palabra válida con menor distancia de Hamming*

Demostración. Siendo N el largo de las palabras del código. Sea r la palabra recibida. Se busca la palabra del código c tal maximice el $\log \mathbb{P}(r/c)$. Sea d la cantidad de bits en que difieren r y c (la distancia de Hamming entre ambas palabras). Por lo tanto,

$$\begin{aligned} \log(P(r/c)) &= \log(p_e^d(1-p_e)^{N-d}) && (1.1)\\ &= d\log(p_e) + (N-d)\log(1-p_e) && (1.2)\\ &= d\log(\frac{p_e}{1-p_e}) + N\log(1-p_e) && (1.3) \end{aligned}$$

y por lo tanto si $p_e < 0{,}5$ se cumple que $\log(\frac{p_e}{1-p_e}) < 0$ lo que concluye el teorema ya que $\max \log \mathbb{P}(r/c) = \min d$ □

Ejercicio 1.2. *a)Encuentre la distancia mínima de los siguientes códigos*

1. *0000, 1100, 1010, 1001, 0110, 0101, 0011, 1111 ;*
2. *10000, 01010, 00001 ;*
3. *000000, 101010, 010101 .*

En cada caso especifique el número de errores que pueden ser detectados y corregidos, y cuál es la eficiencia de cada código.

b)¿Cuáles de los códigos anteriores pueden ser extendidos agregando una palabra más de código sin alterar su capacidad de corregir o detectar errores?

c) Determine la distancia mínima de la función de codificación

$$\begin{aligned} e(000) &= 00000000\\ e(001) &= 01110010\\ e(010) &= 10011100\\ e(011) &= 01110001\\ e(100) &= 01100101\\ e(101) &= 10110000\\ e(111) &= 00001111 \end{aligned}$$

¿Cuántos errores detectará e?

¿Cuántos errores corregirá e?

1.4. Códigos lineales y códigos lineales de bloques

1.4.1. Introducción

En las secciones anteriores se analizó cómo calcular la mínima distancia de Hamming necesaria de un código de acuerdo a la cantidad de errores que se desea que pueda detectar y corregir. El siguiente problema que se plantea es cómo elegir las palabras de código. Por ejemplo si se quieren transmitir mensajes de 4 bits con un código que pueda corregir errores de un bit, son necesarias 16 palabras de código con distancia mínima de Hamming 3 entre todas ellas. La pregunta es cuánto vale el mínimo N que puede cumplir con este requerimiento. El mínimo N vale 7, pero encontrarlo implica una búsqueda exhaustiva para cada caso. Por lo tanto, se hace necesario diseñar técnicas que permitan generar códigos de forma más inteligente. La primera clase de códigos que se estudiará son lo códigos lineales de bloques. A continuación se darán algunas definiciones para caracterizar estos códigos.

Definición 1.3 (Códigos de bloques)**.** Un código de bloques mapea un conjunto de k bits a transmitir (2^k palabras) en un bloque de n bits. Se habla entonces de códigos (n, k).

Los códigos algebraicos realizan el mapeo mediante operaciones algebraicas sobre el bloque de k bits. En un código lineal binario, el mapeo es una función lineal con las operaciones módulo-2. Se transmiten las palabras de n bits y por tanto la tasa de información del código será $\frac{k}{n}$.

Definición 1.4 (Códigos lineales (de bloque o no))**.** Un código lineal binario C de largo n es un conjunto de n-uplas binarias tal que la suma módulo-2 componente a componente de cualesquiera dos palabras de código genera una palabra de código.

Los códigos convolucionales que se verán más adelante también son lineales aunque no de bloques.

Observar que la definición anterior implica que la palabra de todos $0s$ es siempre una palabra de código de un código lineal ¿por qué?.

Ejemplo 1.3. *Por ejemplo el código* $(000, 101, 011)$ *no es un código lineal. ¿por qué? ¿Es posible transformarlo en un código lineal?*

Teorema 1.4. *Si se define el peso de una palabra como el número de* $1s$ *que contiene. La mínima distancia de Hamming en un código lineal es igual al peso de la palabra distinta de cero y con menor peso.*

Demostración. La idea de la prueba se basa en las siguientes observaciones:

1. La suma de dos palabras del código es una palabra del código

2. La distancia de Hamming entre dos palabras es la cantidad de lugares en la que difieren y por tanto si las sumo, la distancia de Hamming es la cantidad de lugares en que la suma tiene un uno.

3. La observación anterior se puede expresar como: $H(w_1, w_2) = peso(w_1 + w_2)$

4. De lo anterior se infiere que la distancia entre dos palabras es siempre el peso de otra palabra del código.

Los detalles de la prueba se dejan al lector. □

1.4.2. Una visión desde el álgebra del problema de codificación

Introducción

Cuando se estudió modulación, se buscó en el espacio de todas las señales, un subespacio tal que las señales que se envían son combinaciones lineales de un conjunto de señales ortogonales que generan dicho subespacio. Luego de transmitidas estas señales por efecto del ruido por ejemplo, pueden dejar de pertenecer a dicho subespacio. Lo que se hace para demodular es encontrar dada la señal recibida la proyección a dicho subespacio y decidir la señal enviada como aquella que está a menor distancia de la proyección.

El problema de la codificación es conceptualmente el mismo. Se desean enviar palabras de n bits para codificar 2^k mensajes. El problema es en el espacio de todos los vectores de n bits (2^n vectores, $n > k$), cómo elegir un conjunto de 2^k vectores. Este conjunto definiría un subespacio vectorial y una vez recibida una palabra se podría proyectar sobre dicho subespacio con la misma idea que lo visto en modulación.

Se analizará a continuación el espacio vectorial de mensajes de n bits a los efectos de evaluar la factibilidad del procedimiento descripto antes y de otros procedimientos que permitan codificar. Esta formalización algebraica de los códigos resulta de utilidad para entender diferentes técnicas que se verán más adelante. También se observará que el producto interno y por tanto las nociones de ortogonalidad y proyección en este espacio, no cumple todas propiedades habituales del producto interno. Por lo tanto, el procedimiento para codificar/decodificar no podrá utilizar las nociones de proyección que estamos habituados a utilizar en otros espacios.

El cuerpo finito $GF(2)$, el espacio vectorial $GF(2)^n$ y los códigos como subespacios vectoriales

Se puede probar sin dificultad que el alfabeto binario {0,1} junto con las operaciones de suma y multiplicación módulo dos son un cuerpo finito. Se recuerda al lector que hablando informalmente un cuerpo es un conjunto de elementos sobre el cual se definen las operaciones de suma, resta, multiplicación y división dando como resultado elementos del propio conjunto. Las operaciones de suma y multiplicación deben ser conmutativas asociativas y distributivas. El cuerpo binario se denomina habitualmente $GF(2)$ (Galois Field de orden 2).

También se puede probar que las n-uplas binarias junto con el cuerpo $GF(2)$ son un espacio vectorial. Este espacio vectorial se nota habitualmente como $GF(2)^n$. A partir de este punto las nociones de independencia lineal y base del espacio vectorial se aplican directamente. Así mismo la noción de subespacio vectorial y la propiedad de que un subconjunto del espacio es un subespacio si toda combinación lineal del vectores del subespacio es cerrada en el subespacio.

Un código lineal (n,k) de mensajes de k bits codificado con palabras de código de largo n, es un subespacio del espacio n dimensional de las palabras de n bits con la aritmética módulo-2 $GF(2)^n$. ¿por qué? Esta observación establece una primera relación entre el álgebra y la teoría de codificación: un código lineal es un subespacio vectorial del espacio $GF(2)^n$. En la próxima sección se analizará como un base de este espacio proporciona una forma simple de codificar un código lineal.

1.4.3. Base de un código lineal binario y matriz generadora

Sea $\mathbf{x} = (x_1, .., x_k)$ uno de los mensajes a transmitir sin codificar. Sea $\mathbf{c} = (c_1, .., c_n)$ la palabra codificada correspondiente. Como el código de bloques lineal y binario cada bit de C se obtiene a través de un mapeo lineal y con operaciones binarias sobre los bits de $\mathbf{x}$, es decir,

$$c_j = x_1 g_{1j} + x_2 g_{2j} + ... + x_k g_j \forall j = 1, ...n.$$

Se puede entonces escribir esa ecuación matricialmente

$$\mathbf{c} = \mathbf{x}\mathbb{G}$$

donde $\mathbb{G}$ se denomina la matriz generadora del código:

$$\mathbb{G} = \begin{pmatrix} g_{11} & ... & g_{1n} \\ ... & ... & ... \\ g_{k1} & ... & g_{kn} \end{pmatrix}$$

Como se mencionó antes, un código (n, k) es un subespacio vectorial del espacio de las palabras de n bits. Por lo tanto, este subespacio vectorial estará especificado por una base de dicho subespacio. Eso lleva a una definición formal de la matriz $\mathbb{G}$:

Definición 1.5 (Matriz generadora de un código C)**.** Es una matriz $\mathbb{G}(k \times n)$ cuyas filas son una base del sub-espacio vectorial definido por el código lineal (n, k), C.

Lo que dice la definición, es que el código C es el subespacio vectorial generado por las filas de $\mathbb{G}$. Evidentemente esto implica que las filas de $\mathbb{G}$ deben ser l.i. y por lo tanto $\mathbb{G}$ es una matriz de dimensiones $k \times n$ y de rango k. Como la base de un subespacio no es un conjunto único de vectores, $\mathbb{G}$ no es única para un código C.

Ejemplo 1.4. *Sea el código* $C = 000, 100, 001, 101$ *¿Es un código lineal de bloques? ¿Es un subespacio vectorial de* $GF(2)^3$*? Se puede responder estas preguntas observando que la suma de dos palabras cualesquiera del código da una palabra del código. ¿Cuál es la dimensión de este subespacio? En este caso se puede ver fácilmente que* $100, 001$ *es una base del subespacio ya que cualquier otra palabra de código se forma como combinación lineal de estas dos. La dimensión del subespacio es 2 y tendrá una matriz generadora de dimensiones* 2×3*. Por lo tanto, si utilizamos esta base la matriz generadora será:*

$$\mathbb{G} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Con esta matriz, a los cuatro mensajes posibles a mandar de dos bits: $00, 10, 01, 11$ *le corresponderán las siguientes palabras codificadas (obtenidas multiplicando por* $\mathbb{G}$ *los mensajes):* $000, 100, 001, 101$*. ¿qué sucede si en lugar de la base anterior utilizo otra base, por ejemplo:* $100, 101$*? ¿es el mismo código? ¿cuál es la diferencia entre usar una base u otra de las anteriores?*

Ejercicio 1.3. *a) Sea C un código lineal definido por la siguiente matriz generadora*

$$\mathbb{G} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

¿Cuál es la dimensión del código? ¿de qué largo son las palabras codificadas? ¿Cuál es la codificación del mensaje de todos $1s$*?*

b) Se necesita generar un código C *de mensajes de dos bits a ser codificados en 3 bits, esto es un subespacio de dimensión dos del espacio* $GF(2)^3$*. Arbitrariamente se elige como base los vectores* $110, 011$*. ¿Cual es la matriz generadora del código? ¿cuáles son las palabras del código?*

Definición 1.6 (Código sistemático)**.** Un código lineal de bloques se dice sistemático si su matriz $\mathbb{G}$ puede escribirse de la siguiente forma:

$$\mathbb{G} = \left[\mathbb{I}_k|\mathbb{P}\right]$$

donde $\mathbb{I}_k$ es la identidad de orden k. $\mathbb{P}$ es una matriz $k \times (n-k)$ que determina los bits de paridad. Observar que:

$$\mathbf{c} = \mathbf{x}\mathbb{G} = \mathbf{x}\left[\mathbb{I}_k|\mathbb{P}\right] = (x_1, .., x_k, q_1, .., q_{(n-k)})$$

donde:

$$q_j = x_1 p_{1j} + .. + x_k p_{kj}$$

La pregunta que surge es si todo código lineal de bloques puede escribirse en forma sistemática. Es decir, si siempre es posible cambiando de base obtener una matriz generadora en forma sistemática y que sus filas generen el mismo código. La respuesta es que no siempre, pero si no posible, siempre existe un código equivalente que tenga representación en forma sistemática.

Definiremos a continuación que se entiende por código equivalente, pero lo importante para entender en qué sentido son equivalentes es que son códigos que mantienen los mismos 3 parámetros: (n, k, d) es decir que utilizan n bits para las palabras de código y son capaces de codificar 2^k mensajes con una distancia mínima de Hamming igual a d.

Definición 1.7 (Códigos equivalentes)**.** Dos códigos binarios son equivalentes si las palabras de uno pueden ser obtenidas de las del otro mediante permutaciones de las posiciones en las palabras de código.

Ejemplo 1.5. *Los siguientes códigos son equivalentes* $C = 000, 101, 010, 111$ *y* $C' = 000, 011, 100, 111$ *ya que uno se obtiene del otro permutando los dos primeros bits.*

Observar que como un código se obtiene del otro permutando posiciones, la distancia mínima del código no se modifica ya que la distancia mínima como se vio antes es igual al peso de la palabra no nula de de menor peso. Al permutar posiciones el peso de cada palabra no cambia.

Teorema 1.5. *Dos matrices* $k \times n$ *generan códigos equivalentes si una matriz puede ser obtenida de la otra mediante las siguientes operaciones:*

1. *Permutación de las filas.*
2. *Combinación lineal de filas*
3. *Permutación de columnas.*

Demostración. Las primeras dos operaciones simplemente cambian de base y generan el mismo código. La segunda convierte una matriz generadora de un código en la matriz generadora de uno equivalente. □

Teorema 1.6. *Dado un código lineal de bloques* (n, k, d) *existe un código equivalente cuya matriz se puede escribir de forma sistemática.*

Demostración. La idea de la prueba es que dada la matriz $\mathbb{G}$ aplicar el método de eliminación Gaussiana para llevarla a su forma en escalera. Si la nueva matriz está en su forma sistemática, el código que se obtiene de esta nueva matriz es el mismo que el original. Si no estuviera en la forma sistemática es fácil ver que se puede llevar a esta forma mediante permutaciones de columnas. □

En el ejercicio 1.3 b) se eligió arbitrariamente la base como $110, 011$, obteniendo como matriz generadora:

$$\mathbb{G} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Se podría haber elegido otro conjunto como base haciendo permutaciones de estos vectores y por ejemplo intercambiando la segunda y la tercera columna de cada vector y se obtendría:

$$\mathbb{G}' = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

que es la forma sistemática de la matriz $\mathbb{G}$.

El representar las matrices generadoras en forma sistemática, hace más simple que la tarea de decodificación. Esto se verá más adelante.

Ejemplo 1.6. *Se considera el código* $(7, 4)$ *donde la matriz generadora del código en forma sistemática es:*

$$\mathbb{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Por lo tanto la palabra de código correspondiente a la palabra a enviar (1101) *es*

$$(1101)\mathbb{G} = (1101001)$$

En este caso el cálculo de los bits de la palabra de código a partir de los bits de información se hace a través de las siguientes ecuaciones $c_0 = x_0$; $c_1 = x_1$; $c_2 = x_2$; $c_3 = x_3$; $c_4 = x_0 + x_1 + x_2$; $c_5 = x_1 + x_2 + x_3$; $c_6 = x_0 + x_1 + x_3$.

1.4.4. Matriz de chequeo de paridad

Definición 1.8 (Matriz de chequeo de paridad $\mathbb{H}$)**.** Sea C un código lineal (n, k). Una matriz $(n - k) \times n$, $\mathbb{H}$, es de chequeo de paridad de C si y solo si $\mathbf{c}\mathbb{H}^t = \mathbf{0}$ para todo $\mathbf{c} \in C$.

Ejemplo 1.7. *Sea* C *el código de repetición de 3 bits* $111, 000$ *entonces decimos que una matriz válida de chequeo de paridad para este código será:*

$$\mathbb{H}^t = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$

ya que $\mathbf{c}\mathbb{H}^t$ *es*

$$\begin{pmatrix} u, v, w \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} = (u + w, v + w)$$

Si las únicas palabras de código posible son $000, 111$*, se cumple que para toda palabra válida* $u = v = w$*, y por lo tanto* $u + w = 0$ *y* $v + w = 0$*.*

Volviendo al álgebra, el lector recordará que el espacio nulo de una matriz $\mathbb{H}$ es el conjunto de vectores $\mathbf{x}$ tales que $\mathbb{H}\mathbf{x}^t = \mathbf{0}$. Por lo tanto, vemos una conexión directa entre el espacio nulo de $\mathbb{H}$ y el código C. Las palabras de código de C forman el espacio nulo de $\mathbb{H}$. Recordar que si $\mathbf{x}$ verifica $\mathbb{H}\mathbf{x}^t = \mathbf{0}$, entonces también verifica $(\mathbb{H}\mathbf{x}^t)^t = \mathbf{x}\mathbb{H}^t = \mathbf{0}$ y viceversa.

Dada la matriz generadora de un código lineal de bloques en su forma sistemática:

$$\mathbb{G} = \left[\mathbb{I}_k | \mathbb{P}\right]$$

La matriz de chequeo de paridad de dicho código tiene la siguiente forma:

$$\mathbb{H} = \left[\mathbb{P}^t | \mathbb{I}_{n-k}\right]$$

Ejemplo 1.8. *Por ejemplo para el código* $(7, 4)$ *y la matriz generadora correspondiente vista en el ejemplo anterior, la matriz de paridad correspondiente será:*

$$\mathbb{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

El lector puede verificar que cumple la definición de matriz de paridad.

Con la definición dada a la matriz de paridad en este caso se verifica que:

$$\mathbb{G}\mathbb{H}^t = \left[\mathbb{I}_k | \mathbb{P}\right] \begin{bmatrix} \mathbb{P} \\ \mathbb{I}_{n-k} \end{bmatrix} = \mathbb{P} + \mathbb{P} = \mathbf{0}$$

$$\mathbb{H}\mathbb{G}^t = \left[\mathbb{P}^t | \mathbb{I}_{n-k}\right] \begin{bmatrix} \mathbb{I}_k \\ \mathbb{P}^t \end{bmatrix} = \mathbb{P}^t + \mathbb{P}^t = \mathbf{0}$$

La última identidad de ambas ecuaciones se debe a que se está trabajando en aritmética módulo dos y por lo tanto la suma de un número con sigo mismo es cero.

Por lo tanto es fácil verificar que definida de esa forma la matriz de paridad a partir de la matriz generadora es realmente una matriz de paridad ya que para toda palabra de código $\mathbf{c}$

$$\mathbf{c}\mathbb{H}^t = \mathbf{x}\mathbb{G}\mathbb{H}^t = \mathbf{0} \tag{1.4}$$

En general cuando la matriz $\mathbb{G}$ no se encuentra en forma sistemática también se puede definir $\mathbb{H}$ como la matriz que cumple que

$$\mathbb{G}\mathbb{H}^t = \mathbf{0}$$

Ejemplo 1.9. *Por ejemplo para el código* $(7, 4)$ *visto antes*

$$\mathbb{G}\mathbb{H}^t = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

El lector puede verificar que cumple la propiedad anterior.

La ecuación 1.4 dice que multiplicando cualquier palabra del código por la matriz de paridad el resultado debe ser el vector nulo. Esto permite en el receptor decidir si hubo errores. Es decir, si se multiplica la palabra recibida por la matriz de paridad y el resultado es diferente de cero esto indica que se produjeron errores. Si da cero podemos asumir que no hubo errores. Esto es cierto salvo que se produjeran tantos errores que una palabra de código se transforme en otra palabra válida del código.

1.4.5. El síndrome y la decodificación

Sea $\mathbf{c}$ la palabra de código transmitida y $\mathbf{r}$ la correspondiente secuencia recibida. Se cumple que:

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

Siendo $\mathbf{e}$ el patrón de errores producido durante la transmisión, es decir la secuencia de bits que tiene un 1 en los lugares donde se modificó un bit. Para corregir errores se observa que si $\mathbf{e}$ puede ser determinado entonces es posible reconstruir la palabra enviada usando:

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

Definición 1.9 (Síndrome)**.** Se define el síndrome $\mathbf{s}$ de la siguiente forma:

$$\mathbf{s} = \mathbf{r}\mathbb{H}^t$$

Se observa que

$$\mathbf{s} = \mathbf{r}\mathbb{H}^t = (\mathbf{c} + \mathbf{e})\mathbb{H}^t = \mathbf{e}\mathbb{H}^t$$

El síndrome depende solo del patrón de error y no de la palabra transmitida y por lo tanto cada patrón de error tiene un síndrome asociado. Hay que hacer notar sin embargo que varios patrones de error pueden conducir al mismo síndrome.

Por otra parte, se debe observar que el síndrome correspondiente a patrones de errores de un solo bit corresponde a las columnas de la matriz $\mathbb{H}$. ¿ por qué? Es más, a partir de la ecuación anterior es fácil probar que el síndrome de una palabra de código recibida es un vector que es igual a la suma de las columnas de H que corresponden a posiciones donde ocurrieron errores.

En general, el sistema de ecuaciones

$$\mathbf{s} = \mathbf{e}\mathbb{H}^t$$

tiene $n-k$ ecuaciones y n incógnitas. Por lo tanto, hay 2^k patrones de error que conducen al mismo síndrome. En general se asume que el patrón mas probable es aquel con peso mínimo.

Ejemplo 1.10. *Se utiliza el código* $(7,4)$ *visto antes, y la palabra recibida es* 1101000.

$$(1101000)\mathbb{H}^t = (1101000)\begin{pmatrix}1&0&1\\1&1&1\\1&1&0\\0&1&1\\1&0&0\\0&1&0\\0&0&1\end{pmatrix} = (001)$$

Al no ser el vector nulo quiere decir que se ha producido al menos un error. Si se produjo un solo error, hay que ver a qué columna de $\mathbb{H}$ *corresponde el síndrome obtenido. En este caso corresponde a la última columna de* $\mathbb{H}$*. Por lo tanto, el patrón de error* $\mathbf{e} = (0000001)$ *y por lo tanto la palabra enviada más probablemente fue* 1101001*. Observar que este resultado es consistente con lo visto en el ejemplo* 1.6*. En segundo lugar se puede observar que todos los patrones de error que verifiquen el siguiente sistema de ecuaciones darán el mismo síndrome :*

$$\mathbf{e}\mathbb{H}^t = \begin{pmatrix}e_0+e_1+e_2+e_4=0\\e_1+e_2+e_3+e_5=0\\e_0+e_1+e_3+e_6=1\end{pmatrix}$$

Así por ejemplo $(0,0,0,1,0,1,0)$ *que tiene 2 errores también daría el mismo síndrome.*

Ejercicio 1.4. *a) Sea* C *un código lineal definido por la siguiente matriz de paridad* H

$$\mathbb{H} = \begin{pmatrix}1&1&0&1&0&1\\1&1&0&0&1&0\\1&0&1&1&0&0\end{pmatrix}$$

Si se recibe la palabra 110110 y solo se comete un error ¿cuál es la palabra de código enviada?

b) Considere un código $(7,4)$ *con matriz generadora*

$$\mathbb{G} = \begin{pmatrix}0&1&0&1&1&0&0\\1&0&1&0&1&0&0\\0&1&1&0&0&1&0\\1&1&0&0&0&0&1\end{pmatrix}$$

(b.1) Encuentre todas las palabras del código.
(b.2) ¿Cuál es la distancia mínima del código?
(b.3) Encuentre la matriz de paridad H *del código.*

(b.4) Encuentre el síndrome del vector recibido R = [1101011]. ¿Qué conclusiones puede extraer respecto de la secuencia enviada?

(b.5) Usando operaciones sobre las filas y las columnas lleve G a su forma sistemática y encuentre la matriz de paridad H correspondiente. Bosqueje una implementación como un filtro de este código.

Por último enunciaremos dos teoremas cuya prueba puede verse en [5].

Teorema 1.7. *La mínima distancia de un código lineal de bloques es igual al mínimo peso de sus palabras no nulas y viceversa.*

Teorema 1.8. *Sea C un código lineal (n, k) con matriz de paridad $\mathbb{H}$.Por cada palabra de código de peso l existen l columnas de $\mathbb{H}$ tales que el vector suma de estas l columnas es igual al vector nulo. En el otro sentido, si existen l columnas de $\mathbb{H}$ cuyo vector suma es el nulo, existe una palabra de código de peso l en C.*

Estos Teoremas permiten evaluar la distancia de Hamming de un código a partir de observar las palabras de código o la matriz $\mathbb{H}$.

1.4.6. OPCIONAL: El problema de decodificación y la proyección ortogonal

Antes de estudiar algunos tipos particulares de códigos lineales de bloques, en esta sección se analizará el problema de la proyección y la ortogonalidad en el espacio $GF(2)$. En el espacio vectorial $GF(2)^n$ se puede definir el producto escalar de dos vectores:

Definición 1.10 (Producto escalar de dos vectores de $GF(2)^n$)**.** Sean $\mathbf{u} = (u_1, .., u_n)$ y $\mathbf{v} = (v_1, .., v_n)$ dos vectores de $GF(2)^n$. El producto escalar de ellos $\mathbf{u}.\mathbf{v} = u_1v_1 + .. + u_nv_n$, donde la suma y la multiplicación son las habituales en $GF(2)$.

Definición 1.11 (Ortogonalidad en $GF(2)^n$)**.** Dos vectores $\mathbf{u} = (u_1, .., u_n)$ y $\mathbf{v} = (v_1, .., v_n)$ son ortogonales si $\mathbf{u}.\mathbf{v} = 0$

Definición 1.12 (Subespacios ortogonales en $GF(2)^n$)**.** Dado un subespacio de S de $GF(2)^n$, el espacio de todos los vectores ortogonales a S es llamado el complemento ortogonal de S y se nota $S^{\perp}$

Se debe observar que dada una matriz, el subespacio nulo es complemento ortogonal del subespacio generado por las filas de la matriz. Esto es fácil de observar ya que las filas de la matriz contienen una base de este subespacio y el subespacio nulo es el de todos los vectores ortogonales a todas las filas de la matriz.

Definición 1.13 (Código dual)**.** Dado un código lineal (n, k) , C, el código dual de C notado $C^{\perp}$, es el formado por todos los vectores de $GF(2)^n$ ortogonales a toda palabra de código de C.

Por lo tanto, los conceptos de código dual y complemento ortogonal son los mismos. En este punto el lector podría aventurarse y decir que si se tiene la noción de ortogonalidad y de subespacio ortogonal, es posible proyectar y utilizar la distancia generada por el producto interno definido para decodificar.

Sin embargo, este camino es incorrecto. Si bien se ha definido un producto escalar entre vectores, este producto no cumple una propiedad básica para ser realmente un producto interno y definir una norma:

$\mathbf{u}.\mathbf{u} = 0$ si y solo si $u = 0$.

Se debe observar que $\mathbf{u}.\mathbf{u} = 0$ para todo vector $\mathbf{u}$ que contenga un número par de unos. Esto tiene consecuencias también sobre la noción de ortogonalidad en este espacio. Hay muchos vectores que son ortogonales a si mismos. Por lo tanto, en este espacio un subespacio y su subespacio ortogonal no son disjuntos (a menos del vector nulo) como es habitual en R^n por ejemplo.

Por este motivo, la decodificación es un proceso que se hace a través de la matriz $\mathbb{H}$ y el estudio del síndrome como ya se ha estudiado. Sin embargo, las nociones de ortogonalidad están presentes en el proceso de decodificación ya que se puede probar que:

Teorema 1.9. *Sea C un código lineal, $\mathbb{G}$ su matriz generadora y $\mathbb{H}$ su matriz de paridad. Sea $C^\perp$ el código dual de C. Entonces se verifica que:*

1. $\mathbb{H}$ es la matriz generadora de $C^\perp$, es decir que $C^\perp$ es el espacio generado por las filas de $\mathbb{H}$.

2. C es el espacio nulo de $\mathbb{H}$.

3. Las filas de $\mathbb{G}$ son ortogonales con las filas de $\mathbb{H}$.

La matriz $\mathbb{H}$ permite encontrar qué vectores son ortogonales al complemento ortogonal del código C. El código ortogonal al complemento ortogonal de C es decir $(C^\perp)^\perp$ es fácil verificar que es igual a C. Por lo tanto, si es ortogonal al complemento ortogonal de C es una palabra de código. En caso contrario hay un error y que se analizará utilizando el síndrome y la palabra más probable (con menor cantidad de errores) que me puede generar dicho síndrome.

Ejemplo 1.11. *Para terminar y como ejemplo interesante para ver las particularidades de este espacio le proponemos al lector verificar que el código generado por la matriz*

$$\mathbb{G} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

es ortogonal a si mismo. Es decir su código dual es el mismo código.

1.4.7. Códigos de Hamming

Los códigos de Hamming son un tipo particular de los códigos lineales de bloques. Están diseñados para corregir errores de un solo bit. El código de Hamming (n, k) se define a través de un parámetro m y verifica las siguientes propiedades:

- Largo del bloque: $n = 2^m - 1$
- Número de bits de datos: $k = 2^m - m - 1$
- Número de bits de paridad: $n - k = m$
- Distancia mínima de Hamming del código: $d_{min} = 3$

Una propiedad importante para su utilización es que la matriz $\mathbb{H}$ tiene como columnas todos los posibles vectores binarios de largo m (excepto el vector nulo). Chequear esta propiedad en el código $(7, 4)$ que se ha utilizado como ejemplo en esta sección.

1.4.8. Códigos cíclicos

Los códigos cíclicos son también un tipo particular de códigos lineales de bloques.

Definición 1.14 (Códigos cíclicos)**.** Un código lineal de bloques (n, k) es llamado código cíclico si cada desplazamiento cíclico de un vector de código $\mathbf{c}$ es también un vector de código.

Esta estructura particular hace que sea más simple su codificación y decodificación. Cuando se trabaja con códigos cíclicos es conveniente asociar a los vectores del código $\mathbf{c} = (c_0, ..c_{n-1})$ de largo n, a polinomios de grado menor o igual que n cuyo coeficientes son las componentes del vector $c(X) = c_0 + c_1X + ... + c_{n-1}X^{n-1}$.

Entre los vectores y los correspondientes polinomios existe una relación uno a uno. Por este motivo se utilizará en adelante los términos palabra de código y polinomio de código de forma intercambiable, como dos representaciones de una misma cosa.

Como se verá más adelante la representación polinómica de la palabra de código tiene ventajas ya que facilitará el proceso de codificación y decodificación.

Ejemplo 1.12. *El polinomio correspondiente a la palabra de código* 1101 *será* $v(X) = 1 + x + x^3$

Si se considera la palabra de código $\mathbf{v} = (v_0, ..v_{n-1})$ y los componentes de $\mathbf{v}$ son cíclicamente corridos hacia la derecha i lugares obtenemos el vector:

$$\mathbf{v}^{(i)} = (v_{n-i}, ., v_0, .., v_{n-i-1})$$

El polinomio correspondiente a este vector será:

$$v^{(i)}(X) = v_{n-i} + v_{n-i+1}X + ... + v_0X^i + .. + v_{n-i-1}X^{n-1}$$

Existe una relación entre $v(X)$ y $v^{(i)}(X)$. Multiplicando $v(X)$ por X^i se obtiene:

$$X^iv(X) = v_0X^i + v_1X^{i+1} + ... + v_{n-1}X^{n+i-1}$$

Operando se obtiene:

$$\begin{aligned} X^iv(X) &= v_{n-i} + v_{n-i+1}X + ... + v_0X^i + .. + v_{n-i-1}X^{n-1} \\ +v_{n-i}(X^n + 1) &+ v_{n-i+1}X(X^n + 1) + .. + v_{n-1}X^{i+1}(X^n + 1) \\ &= q(X)(X^n + 1) + v^{(i)}(X) \end{aligned}$$

De esta ecuación podemos ver que $v^{(i)}(X)$ es el resto que resulta de dividir el polinomio $X^iv(X)$ por $X^n + 1$. Esto implica que si un vector del código se multiplica por X^i módulo $X^n + 1$ se obtiene otro vector de código. Generalizando, si un vector de código se multiplica por cualquier polinomio $a(x)$ binario (módulo $X^n + 1$) se obtiene otro vector de código. Esto es fácil verlo ya que la multiplicación de una palabra de código por cada término del polinomio X^i es otra palabra de código y la suma de los términos del polinomio no nulos es otra palabra de código por ser el código lineal.

Ejemplo 1.13. Sea el código $(0,0,0),(0,1,1),(1,0,1),(1,1,0)$ Se puede ver fácilmente que es un código lineal, la suma de dos de estos vectores da otro vector del código.También que es cíclico ya que cualquier desplazamiento cíclico de un vector del código da otro vector del código.

Los polinomios asociados a los elementos no nulos son : $x+x^2$, $1+x^2$, $1+x$. Si se multiplica $x+x^2$ por x^2 se obtiene x^3+x^4 y esto módulo x^3+1 es $1+x$ que es el polinomio asociado a la palabra del código $(1,1,0)$. Si por ejemplo multiplicamos esa palabra de código $x+x^2$ por el polinomio x^2+1, se obtiene $x+x^2+x^3+x^4$ que módulo x^3+1 es x^2+1, o sea, $x+x^2+x^3+x^4=(1+x)(x^3+1)+x^2+1$

De todos los polinomios asociados a las palabras de un código habrá al menos uno de grado mínimo. En el ejemplo anterior $g(x)=1+x$.

Con las propiedades vistas se puede probar sin mayor dificultad que el polinomio no nulo y de menor grado de un código cíclico $g(x)$:

a) es único. Ya que si hubiera dos de igual grado, su suma sería un polinomio del código (por ser suma de dos y ser lineal el código). Su suma tendrá grado menor, lo cual es absurdo salvo que sea el polinomio nulo.

b) el término que no depende de X es igual a uno.

Es decir el polinomio de grado mínimo es único y tiene la forma:

$g(X)=1+g_1X+..+X^r$

Como se vio antes los corrimientos cíclicos de $g(X)$, es decir $X^ig(X)$ modulo X^n+1, son también palabra de código. Se puede probar que $g(X)$ y todos sus desplazamientos cíclicos son una base generadora del código.

De donde se observa lo siguiente. Sea $g(X)$ el polinomio de grado mínimo de un código cíclico (n,k). Un polinomio de grado $n-1$ o menor es una palabra válida de este código si es múltiplo de $g(X)$.

$g(X)$ se denomina polinomio generador del código cíclico.

Se puede verificar fácilmente que en el ejemplo anterior $1+x^2$ y $x+x^2$ son divisibles por $g(x)=1+x$ que es el polinomio generador de este código.

Una palabra de código $c(X)$ es una palabra válida de un código cíclico generado por el polinomio $g(X)$ si y solo si $g(X)$ divide a $c(X)$, es decir :

$$\frac{c(X)}{g(X)}=q(X)$$

Se puede probar también que en un código cíclico (n,k) el polinomio generador tiene grado $n-k$: $g(X)=1+g_1X+...+X^{n-k}$ y como $g(X)$ divide a toda palabra de código, entonces se puede expresar la palabra de código de la forma:

$c(X)=(u_0+u_1X+..+u_{k-1}X^{k-1})g(X)$

Por lo tanto si los coeficientes de u corresponden a los k dígitos de información a codificar, la palabra de código se obtiene multiplicando el polinomio correspondiente a la palabra a codificar por el polinomio generador.

Los códigos cíclicos se utilizan muchas veces para detección de errores por su facilidad de codificación y decodificación. Los códigos de redundancia cíclicos (CRC) son muy utilizados para este fin. Existen varios códigos CRC estandarizados. Por ejemplo 802.11 utiliza para detectar errores en las tramas el código CRC-32 cuyo polinomio generador es:

$g(X)=X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$

Ejercicio 1.5. *a)¿Cuáles de los siguientes códigos son cíclicos?*

1. *000, 100, 010;*
2. *000, 100, 010, 001;*
3. *000, 111;*
4. *0000, 1010, 0101, 1111.*

b) El código de Hamming (15,11) tiene como polinomio generador $g(X) = 1 + X + X^4$*. Determinar si las siguientes palabras de código son válidas* $c_1(X) = 1 + X + X^3 + X^7$ *y* $c_2(X) = 1 + X^3 + X^5 + X^6$

d) El código de Hamming cíclico (7,4) tiene un polinomio generador $g(X) = 1 + X^2 + X^3$ *.*

(d.1) Encontrar la matriz generadora del código.

(d.2) Encontrar la matriz de chequeo de paridad.

(d.3) Suponga que la palabra de código C = [1011010] es transmitida y la palabra correspondiente recibida es C = [1010011]. Encuentre el síndrome asociado con esta palabra.

(d.4) Encuentre todas las posibles palabras de código recibidas tal que si se transmite C = [1011010], la palabra de código recibida tenga síndrome 0.

Antes de finalizar es importante resaltar una última propiedad que permite codificar cuando se utiliza un código cíclico. Si se quiere codificar la siguiente palabra:

$u(X) = u_0 + u_1X + ... + u_{k-1}X^{k-1}$.

Si se multiplica $u(X)$ por X^{n-k} y se lo divide por el polinomio generador del código $g(X)$ se obtiene:

$X^{n-k}u(X) = a(X)g(X) + b(X)$

donde a y b son el cociente y resto. Como $g(X)$ es de grado $n-k$ el grado de $b(X)$ será $n-k-1$ o menos. Por lo tanto, reordenando se obtiene el siguiente polinomio de grado $n-1$ o menor

$b(X) + X^{n-k}u(X) = a(X)g(X)$

que es múltiplo del polinomio generador y por tanto $b(X) + X^{n-k}u(X)$ es una palabra de código válida que corresponde a la palabra de código:

$(b_0, b_1, ..., b_{n-k-1}, u_0, u_1, ..., u_{k-1})$.

que corresponde a los dígitos de la palabra sin codificar seguido de los bits de paridad. Hemos de esta forma llevado el código a su forma sistemática. Además esto brinda una forma simple de codificar. Se toman los digitos de mensaje se los multiplica por X^{n-k} se divide por el polinomio generador y se envían los digitos del mensaje seguidos del resto de la división.

1.5. Códigos Convolucionales

1.5.1. Introducción

Los códigos de bloques toman k bits y producen n bits de salida, donde k y n son habitualmente 'grandes'. Por otro lado no hay dependencia entre bloques, cada

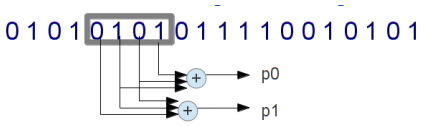

Figura 1.3: Diagrama de la ventana deslizante de un código convolucional .

bloque se codifica independientemente de la información que hubiere en los bloques anteriores y posteriores.

Por el contrario, los códigos convolucionales, que más adelante se verá como se definen, parten de los bits de información y generan un número "pequeño" de bits de manera tal que los datos pasan por el codificador como un flujo continuo de bits. Por esta razón, son muy útiles en comunicaciones porque generan pocos retardos ya que no hay que esperar que llegue todo un bloque para decidir si hay o no errores y eventualmente corregirlos. Otro aspecto interesante de los códigos convolucionales es que se enviarán solo 'los bits de paridad', en el sentido que los datos de información como tales no se transmiten.

Los códigos convolucionales son un tipo de código de corrección de errores lineal que utiliza una ventana deslizante de L bits (L se denomina la restricción del código). Mediante operaciones lineales de los datos que se encuentran en la ventana en cada momento se generan r bits de paridad que son los datos que se envían al transmisor. En la figura 1.3 se muestra un ejemplo de la forma de operación de un código convolucional. Se debe observar que la tasa de información de un código convolucional será $\frac{1}{r}$, ya que por cada bit nuevo que ingresa en la ventana deslizante se sacan r bits de paridad que son los que se mandan. En el ejemplo de la figura 1.3 la tasa de información del codificador es $\frac{1}{2}$.

1.5.2. Formas de representación de un código convolucional

Existen varias formas de representar la operación de un código convolucional. La primera de ellas se muestra en la figura 1.4 y es representar al código convolucional como un shift register. Los datos anteriores al actual son almacenados en un shift register de largo $L-1$ y sobre el dato actual y los datos del shift register se realizan las operaciones para calcular los bits de paridad. En el ejemplo de la figura 1.4, se almacenan los dos datos anteriores y los bits de paridad se calculan como la suma del bit actual y cada uno de los bits almacenados.

Otra forma de representar un código convolucional es a través de un diagrama de estados. La cantidad de estados será 2^{L-1}. Cada transición implica la llegada de un nuevo bit y el nuevo estado será aquél en el que queda el shift register luego del arribo de este nuevo bit. Además en cada transición se indican los r bits de paridad que se generan estando en el estado de partida y habiendo llegado un nuevo bit. En la figura 1.5 se muestra esta representación para el mismo código representado en la figura 1.4 como shift register.

Una última forma de representar un código convolucional (y que será útil en el proceso de codificación que se verá más adelante) es a través del diagrama de Trellis.

El diagrama de Trellis es una representación de diagrama de estados pero donde se agrega además el eje temporal. Esta representación se realiza para una secuencia

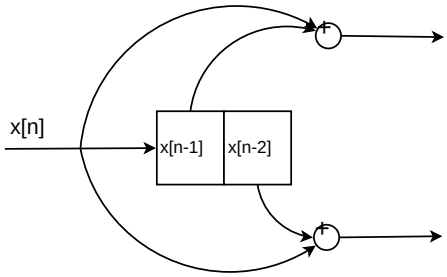

Figura 1.4: Diagrama como shift register de un código convolucional .

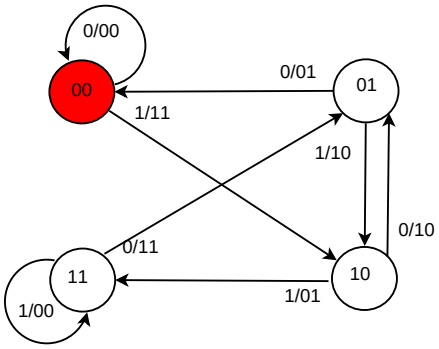

Figura 1.5: Diagrama de estados de un código convolucional .

de entradas y por tanto, permite ver por los estados por los que va pasando el código y las salidas que genera de acuerdo a las entradas que recibe. Por ejemplo en la figura 1.6 se muestra el mismo código que el usado en las figuras 1.5 y 1.4. En este caso para la secuencia de entrada: $0, 1, 0, 0, 1$, el código pasa por los estados $00, 00, 10, 01, 00, 10$ y genera las salidas $00, 11, 00, 01, 11$.

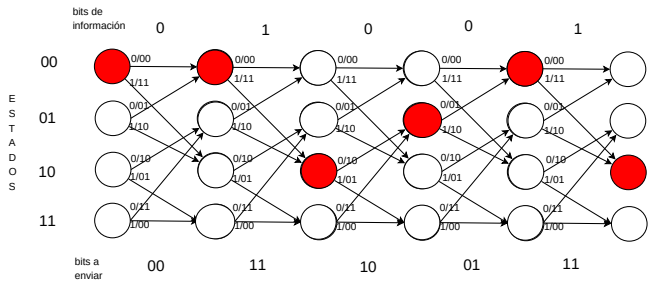

Figura 1.6: Diagrama de Trellis .

1.5.3. OPCIONAL: El código convolucional como un sistema lineal invariante en el tiempo

El codificador de un código convolucional involucra operaciones de retardo y las salidas son combinaciones lineales (en módulo-2) de la señal de entrada retardada. Es por lo tanto un sistema lineal invariante en el tiempo y la salida puede ser obtenida mediante la convolución de la secuencia de entrada y la respuesta al impulso. Para obtener la respuesta al impulso, se pone como entrada el vector $\mathbf{u} = (1, 0, 0, ...)$ y se observan las salidas. Si el codificador es de orden L, la señal debe terminar a lo sumo en $L+1$ unidades de tiempo. Si el codificador tiene M salidas entonces se tendrá una respuesta al impulso de la siguiente forma:

$\mathbf{g}^{(i)} = (g_0^{(i)}, ..g_L^{(i)})$ con $i = 0..M-1$

Las respuestas al impulso $g^{(i)}$ se denominan secuencias generadoras del codificador.

Podemos a partir de la respuesta al impulso escribir las ecuaciones de codificación utilizando el producto de convolución discreto módulo-2:

$\mathbf{v}^{(0)} = \mathbf{u} * \mathbf{g}^{(0)}$

.....

$\mathbf{v}^{(M-1)} = \mathbf{u} * \mathbf{g}^{(M-1)}$

El producto de convolución implica para cada componente n del vector v, la siguiente operación:

$v_n^{(i)} = \sum_{j=0}^{j=L} u_{n-j} g_j^{(i)}$

donde u_{n-j} es cero para $n < j$.

Ejemplo 1.14. *Para el código convolucional de la figura* 1.4, *se tiene que* $M = 2$ *la respuesta al impulso será*

$\mathbf{g}^{(0)} = (1, 1, 0)$ y $\mathbf{g}^{(1)} = (1, 0, 1)$ *Aplicando la ecuación de convolución para dicho código convolucional las salidas serán:* $v_n^{(0)} = u_n + u_{n-1}$ y $v_n^{(1)} = u_n + u_{n-2}$

Luego de ser codificadas las M secuencias de salida son multiplexadas en una sola secuencia denominada palabra de código. La palabra de código viene dada por:

$\mathbf{v} = (v_0^{(0)} v_0^{(1)} .. v_0^{(M-1)}, v_1^{(0)} v_1^{(1)} .. v_1^{(L-1)}, ...)$

El hecho que la secuencia de salida sea una convolución, sugiere el uso de una notación polinómica donde la convolución se transforme en producto.

Definiremos entonces las series de potencia $u(D) = \sum_k u_k D^k$, $g^i(D) = \sum_k g_k^i D^k$ y $v(D) = \sum_k v_k^i D^k$. Estas son llamadas "Transformadas-D" de las secuencias. En estas ecuaciones D no es más que una variable sin embargo puede ser pensado como un operador de retardo, observando que si la transformada-d de $\mathbf{g}$ es $g(D)$, entonces la transformada-D de $D\mathbf{g}$ es $Dg(D)$.

Estas expresiones aparecen totalmente análogas a la transformada-Z sustituyendo D por z^{-1}. La diferencia principal que hay que tener en cuenta es que en la transformada-z, z es una variable compleja y la transformada toma valores sobre los complejos y tiene una interpretación en el dominio de la frecuencia. En cambio en este caso la transformada-D sigue estando en el dominio temporal. Es fácil ver que con esta definición de transformada-D la convolución de dos secuencias se transforma en el producto de los polinomios correspondientes a las transformadas-D de cada secuencia.

Así entonces se obtiene:

$v^i(D) = u(D)g^i(D)$

Se puede escribir la palabra de código entonces como:
$\mathbf{V}(D) = v^i(D)/i \in [0, L-1]$
o luego de multiplexarlos
$\mathbf{v}(D) = v^0(D^L) + Dv^1(D^L) + D^2v^2(D^L) + .. + D^{L-1}v^{L-1}(D^L)$

Ejemplo 1.15. *Para el ejemplo de la figura 1.4, se tiene que:*
$\mathbf{g}^{(0)}(D) = 1 + D$ y $\mathbf{g}^{(1)}(D) = 1 + D^2$ *Si la secuencia de información enviada es* $(1, 0, 1, 1, 1)$ *entonces:*
$\mathbf{u}(D) = 1 + D^2 + D^3 + D^4$ *y por lo tanto:*
$\mathbf{v}^{(0)}(D) = (1 + D^2 + D^3 + D^4)(1 + D) = 1 + D + D^2 + D^5$
y
$\mathbf{v}^{(1)}(D) = (1 + D^2 + D^3 + D^4)(1 + D^2) = 1 + D^3 + D^5 + D^6$
y por lo tanto,
$\mathbf{V}(D) = [1 + D + D^2 + D^5, 1 + D^3 + D^5 + D^6]$
$\mathbf{v}(D) = 1 + D^2 + D^4 + D^{10} + D + D^7 + D^{11} + D^{13} = 1 + D + D^2 + D^4 + D^7 + D^{10} + D^{11} + D^{13}$

Ejemplo 1.16. Verificar para el caso de la figura 1.4 que se obtiene la misma salida que el ejemplo anterior a la entrada $(1, 0, 1, 1, 1)$, haciendo la convolución con la respuesta al impulso.

Con la representación en el dominio de la transformada-D es posible también construir matrices generadoras $\mathbb{G}(D)$ que permiten caracterizar el código y obtener propiedades de él. No nos detendremos en ese tema en estas notas pero el lector interesado puede consultar por ejemplo el libro [5].

1.5.4. Decodificación de un código convolucional

El problema de decodificar un código convolucional no es simple. La primera idea sería utilizar la 'fuerza bruta'. Esto quiere decir que una vez recibida una secuencia se busque dentro del conjunto de secuencias posibles generadas por el código cuál de estas se encuentra a menor distancia de Hamming de la secuencia recibida. Evidentemente este es un trabajo que implica un esfuerzo de cómputo y tiempo muy importante. Afortunadamente a Viterbi se le ocurrió un algoritmo que permite hacer de forma eficiente la decodificación de un código convolucional. Este algoritmo se basa en el diagrama de Trellis que ya se vio antes que representa la secuencia de estados y las salidas generadas por las que pasa un código dada una secuencia de entrada. La idea ahora es ver el problema de manera inversa. Es decir, dada una salida (la secuencia recibida) la idea es encontrar la secuencia de estados y entradas que es más probable que hubiere generado la salida dada. En este contexto la secuencia de estados y de entradas más probable quiere decir aquella que genere una salida que se encuentre a menor distancia de Hamming de la secuencia recibida.

Se explicará en primer lugar sobre el diagrama de Trellis el algoritmo de Viterbi y luego de haber comprendido su operación, se verán algunas consideraciones sobre dicho algoritmo.

La idea del algoritmo de Viterbi es ir calculando la distancia de Hamming entre la palabra recibida y todas las secuencias posibles de bits que pudieran haber sido enviadas. Esto último quiere decir: todos los caminos posibles en el diagrama de Trellis

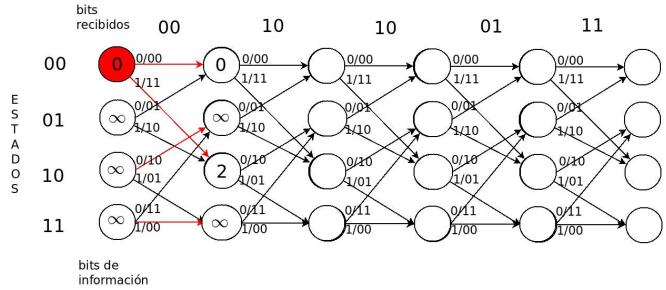

Figura 1.7: Algoritmo de Viterbi.

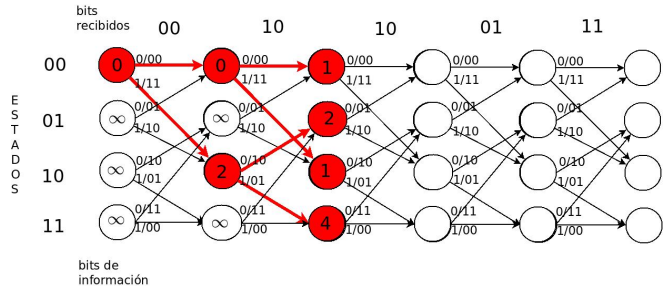

Figura 1.8: Algoritmo de Viterbi.

con el largo del bloque. Lo que aporta el algoritmo de Viterbi es que no es necesario considerar todos los caminos posibles y plantea una forma de ir descartando caminos.

Se asume un estado inicial conocido para cada secuencia, en este ejemplo se asumirá que es el estado 00. Por lo tanto, la distancia al 00 se inicializa en 0 y en ∞ la de todos los demás estados, como se muestra en la figura 1.7. En esta figura se muestra en la parte superior la secuencia de bits recibidos (agrupados de a dos porque cada transición de estados genera dos bits). Observar que respecto a la secuencia enviada (ver figura 1.6) hay un cambio en solo un bit.

Los primeros dos bits recibidos fueron 00. Partiendo del estado 00 puedo ir al estado 00 y la distancia acumulada es 0, es decir 0 del estado anterior más 0 ya que en esa transición se generaría 00 como salida. La otra posible transición desde 00 es al estado 10. En este caso como se muestra en la figura 1.7 la distancia de Hamming acumulada, del estado 10 será 2 ya que en esa transición se hubiera generado 11 como salida y se recibió 00. Los otros dos estados tendrán distancia acumulada ∞ ya que parten de estados con esta distancia. Por lo tanto, los caminos que parten de los estados que no son el 00 se pueden descartar.

En el segundo paso como se muestra en la figura 1.8, se recibió la secuencia 10. De los dos estados con distancia finita se puede pasar a los cuatro estados, con las distancias acumuladas que se muestran en la figura 1.8. Por lo tanto, en este momento se tienen cuatro caminos posibles. Observar que no se pueden descartar los que generan estados con mayor distancia porque más adelante los caminos pueden tener distancias diferentes y la distancia acumulada variar de un camino a otro.

En el tercer paso, como se muestra en la figura 1.9, se recibió la secuencia 10. Se

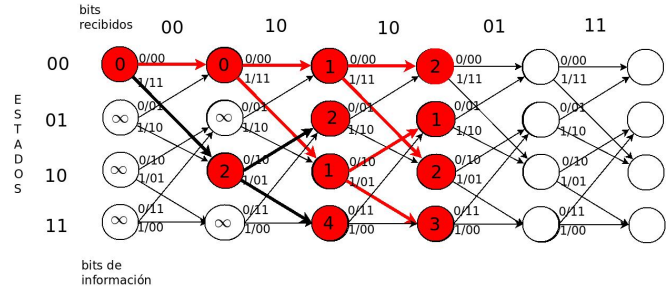

Figura 1.9: Algoritmo de Viterbi.

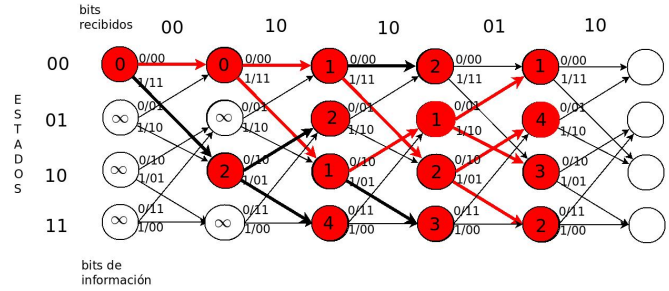

Figura 1.10: Algoritmo de Viterbi.

analizará cada uno de los estados destino en la tercera transición. Al estado destino 00 se puede llegar desde el estado 00 o desde el estado 01. Si se llega desde el estado 00 se tendrá una distancia acumulada 1 (del estado origen) más 1 porque se generaría en esta transición la secuencia 00 y se recibió la 10. Desde el estado 01 se llegaría al estado 00 con distancia acumulada 4, 2 del acumulado del estado anterior más 2 porque se generaría 01 y no 10 como se recibió. Esto permite eliminar el camino que pasa por los estados 00, 10, 01, 00 porque su distancia acumulada termina en el estado 00 pero con distancia acumulada mayor que la del camino 00,00,00,00. Es decir, si un camino $\mathbb{C}_1$tiene mayor distancia acumulada hasta un estado que otro $\mathbb{C}_2$, el de mayor distancia puede ser eliminado porque a partir de allí todos los caminos posibles $\mathbb{C}_i$ tendrán la misma distancia y por tanto la distancia total de $\mathbb{C}_1 + \mathbb{C}_i$, será mayor que la de $\mathbb{C}_2 + \mathbb{C}_i$. De esta forma, en el paso 3 se queda nuevamente solo con 4 caminos posibles.

El paso 4 se muestra en la figura 1.10 al igual que el paso 5, la última transición, que se muestra en la figura 1.11. En esta última figura se llega al final y allí se decide cual es el camino de menor distancia acumulada. Una vez decidido esto, como se muestra en la figura, se obtiene volviendo hacia atrás la secuencia más probable de bits de información que pueden haber generado la secuencia recibida.

Ejercicio 1.6. *a) Considere el determine el código convolucional definido por la función de transferencia en transformada-D:* $\left[1, 1 + D + D^2, 1 + D^2\right]$. *Este es un código de tasa 1/3.*

a.1) Dibuje un diagrama tipo shift-register y un diagrama de estados de este código.

a.2) Dibuje un diagrama de Trellis de 4 transiciones de estados del código anterior.

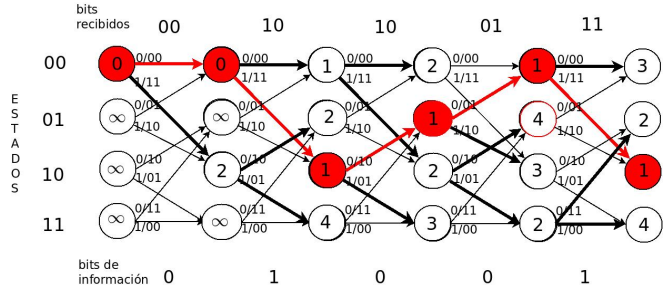

Figura 1.11: Algoritmo de Viterbi.

a.3) Utilizando el algoritmo de Viterbi encuentre la secuencia de entrada más probable si la secuencia de salida recibida es 111,100,001,011.

b) Opcional: Considere el código convolucional generado por el codificador mostrado en la Figura 1.12.

(b.1) Bosqueje el diagrama de Trellis del código.

(b.2) Si se recibe R=[001010001] encuentre la palabra de información y la palabra codificada más probable que se puede haber enviado. ¿Cuál es la distancia de Hamming?

(b.3) Encuentre un camino que se encuentre a distancia mínima de Hamming del camino de todos 0s y calcule dicha distancia.

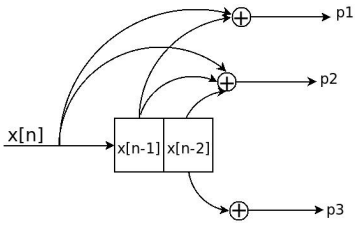

Figura 1.12: Diagrama del código.

1.5.5. Algunas consideraciones de implementación de Viterbi

Muchas veces las secuencias que se envían son un flujo continuo o son secuencias largas de símbolos o bits. El algoritmo de Viterbi necesita almacenar la información del camino más probable para cada estado además de la métrica acumulada para cada estado. El camino más probable crece a medida que la secuencia crece y por razones de memoria y de retardos en general en la práctica las secuencias se truncan. Existen varias técnicas que se utilizan para truncar las secuencias, de todas ellas que se resaltarán dos:

1. Establecer un valor de retardo o de almacenamiento máximo. Cuando por primera vez se alcanza este valor de almacenamiento máximo se saca la primera salida de Viterbi correspondiente a los bits más antiguos recibidos. Esto se hace luego secuencialmente por cada nuevo símbolo recibido y se va obteniendo la salida con el retardo correspondiente al valor de almacenamiento máximo. El problema es que cuando se llega a este valor máximo la secuencia que se saca no sea la de máxima verosimilitud sino que puede ser una secuencia más probable a este punto pero que en el futuro podría cambiar. Se puede probar que si el retardo es suficientemente grande con alta probabilidad la secuencia que se obtiene corresponde a la de máxima verosimilitud.

2. Otra alternativa es al final de cada secuencia de largo fijo enviada se lleva el codificador al estado nulo adicionando bits ficticios al final de la secuencia a enviar. Esto permite que con alta probabilidad se obtenga la secuencia enviada de máxima verosimilitud ya que el decodificar sabe que el codificador utilizó esta técnica y utiliza esta información al decodificar. El inconveniente que tiene es se está bajando la eficiencia del código ya que se están enviando bits que no son de información por el canal.

Cuando se envía tramas de largo fijo en general se las termina también agregando bits adicionales para que el codificador retorne al estado cero. Esto se hace porque en caso contrario los primeros bits de la trama tienen menor probabilidad de error que los últimos. Se debe observar que los primeros bits se utilizaron para obtener las salidas del codificador desde que llegaron al shift register hasta que salieron de él. En cambio, si no se completa con bits adicionales, el último bit de la trama solo se utilizará una vez al arribo al shift register.

1.5.6. Decodificadores hard y soft

El decodificador de Viterbi que se ha descrito en la sección anterior es del tipo decodificador hard. Esto quiere decir que el algoritmo actúa sobre los bits y no sobre los símbolos recibidos. Se podría decodificar sobre los símbolos recibidos y no sobre los bits. Por ejemplo si se supone que los dos bits de salida de un codificador convolucional de tasa $\frac{1}{2}$ se modula utilizando QPSK. En el receptor, cuando se recibe un símbolo en el plano (I, Q), se podría usar la misma lógica que se vio en la sección anterior con el diagrama de Trellis y el decodificador de Viterbi, pero ahora tomando ya no la distancia de Hamming sino la distancia euclídea entre el símbolo recibido y los cuatro símbolos posibles que se podrían haber transmitido. Veremos en la sección siguiente como funciona este mecanismo en Gnuradio.

1.6. Corrección de errores en Gnuradio

Debajo el directorio de instalación de Gnuradio, en el subdirectorio $/gr-trellis$, se encuentran diferentes codificadores y decodificadores que utilizan la representación de Trellis para codificar y decodificar. La máquina de estado es un parámetro de estos algoritmos y la clase que define la máquina de estados está especificada en $/gr-trellis/src/lib/fsm.cc$ y fsm.h.

Estas máquinas de estados se pueden especificar en archivos con un formato que se verá a continuación. En el directorio, $/gr-trellis/src/examples/python/fsm\_files$,

existen varios archivos donde están definidas diferentes máquinas de estado de diferentes codificadores/decodificadores.

Estos archivos tienen el siguiente formato:

La primer línea tiene 3 números que representan: la cantidad de posibles símbolos de entrada (2 si es binario el alfabeto), la cantidad de posibles estados, y la cantidad de posibles salidas.

Luego, se especifica en cada línea cada transición de estados. Para esto el estado actual está implícito en las filas (0,1,2,3 para cuatro estados) y las columnas corresponden también de manera implícita a los diferentes símbolos que pueden llegar (0,1 para un alfabeto binario) y el valor que se muestra es el nuevo estado para cada transición.

Por último, se especifica en cada estado y dependiendo del símbolo que llegue la salida que se tendrá (0,1,2,3 si saca dos bits por cada transición),

Tanto los estados como las salidas se representan en decimal. Por ejemplo la siguiente especificación del archivo $awgn1o2\_4.fsm$, que se encuentra en el directorio mencionado :

2 4 4

0 2
0 2
1 3
1 3

0 3
3 0
1 2
2 1

dice que es un alfabeto binario con cuatro estados y salida de dos bits por cada transición (4 salidas posibles). La primera línea de las transiciones de estado dice que estando en el estado 0 si llega un 0 paso al estado 0 y si llega un 1 paso al estado 2. La primera línea de las salidas generadas dice que estando en el estado 0 si llega un 0 se genera la salida 0 y si llega un 1 se genera la salida 3. Esta máquina de estado corresponde a un código convolucional de tasa $\frac{1}{2}$ con las siguientes salidas: $p_1 = 1+D^2$ y $p_2 = 1 + D + D^2$.

En gnuradio-companion existen varios bloques que permiten implementar codificadores y decodificadores. Varios de ellos son algoritmos de codificación decodificación específicos.

En el curso se utilizará un bloque genérico denominado Trellis-encoder. Este bloque cuya especificación se muestra en la figura 1.13, tiene además de los parámetros básicos (el ID y el tipo), dos parámetros específicos. El primero, denominado FSM args, permite especificar la máquina de estados que se utilizará para codificar, en este parámetro va la ruta y el nombre del archivo con la especificación de la máquina de estados. El segundo parámetro es el estado inicial.

Existen otros tres bloques para decodificar utilizando el algoritmo de Viterbi. El bloque Viterbi, tiene asociada la máquina de estado, la misma que se utilizó para codificar. Esta máquina define además de las transiciones de estado las salidas correspondientes a cada entrada. Pero estas las salidas en la máquina de estados están identificadas por un entero de 0 al número de salidas. Para poder utilizar el algoritmo

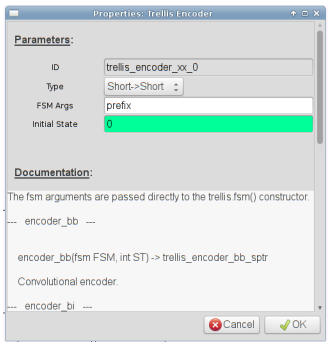

Figura 1.13: gnuradio-companion Trellis encoder.

de Viterbi, con esta especificación de las salidas no es suficiente ya que el algoritmo requiere saber cada salida a qué conjunto de bits o símbolos corresponde para poder calcular las distancias requeridas por el algoritmo. Por este motivo, estas distancias le son suministradas externamente al bloque. La entrada de este bloque son floats (las distancias entre el símbolo o conjunto de bits recibido y los símbolos o conjunto de bits de la constelación). Por ejemplo, si la máquina de estados tiene 4 posibles salidas por cada símbolo recibido, debe recibir 4 floats que corresponden a la distancia a los 4 símbolos. Por este motivo, este bloque debe utilizarse con otro bloque en el flujo que calcule estas distancias para cada símbolo recibido. Este bloque es el denominado Trellis Metrics. Este bloque recibe los símbolos y calcula las distancias. La implementación actual del Trellis-Metric de GnuRadio calcula distancias sobre símbolos y no sobre bits. Esto quiere decir que no calcula las distancia de Hamming entre por ejemplo la salida 00 y las posibles salidas 00,01,10 y 11, sino que calcula una distancia entre el símbolo recibido por ejemplo el correspondiente al 00, -1,-j + ruido y los cuatro símbolos posibles. El bloque permite calcular la distancia euclídea entre el símbolo recibido y los símbolos de la constelación o una distancia Hard entre el símbolo recibido y los de la constelación. Esta distancia Hard vale 0 para el símbolo más cercano de la constelación y 1 para los restantes.

El último bloque que se utilizará es el bloque denominado Viterbi Combo. En este bloque la entrada serán símbolos complejos e implementará un decodificador de Viterbi sobre los símbolos, para una máquina de estado que se especificada como argumento al igual que el largo de bloque. Este bloque es la combinación del bloque Trellis Metric y Viterbi descriptos antes. Por alguna razón este bloque "carga" menos el PC por lo que es recomendable utilizar este bloque Viterbi Combo en lugar de los dos anteriores.

Ejercicio 1.7. *a) Analizar el archivo ejercicio_7_a.grc y calcular el BER para los valores de snr= 5,4,3,2,1,0,-1,-2,-3,-4,-5*

b) Analizar el archivo ejercicio_7_b.grc y calcular el BER para los valores de snr= 5,4,3,2,1,0,-1,-2,-3,-4,-5. Graficar el BER de la parte a) y b) y compararlos. Se propone utilizar una fsm de las que vienen con gnuradio awgn1o2_4.fsm. Deberá referir la variable fsm a la ubicación del archivo en su máquina.

En los archivos de las partes a) y b) hay una variable delay que permite ajustar el retardo de los bits para poder restar los bits de entrada y salida y calcular el BER

correctamente. Verifique que cuando el ruido es despreciable el BER sea 0, si no lo es, modifique el valor de delay hasta que lo sea antes de comenzar a realizar las partes a y b. Importante: Como el BER resulta de un promedio demora en modificar su valor una vez que se cambia. Por esto se sugiere que una vez que se modifica en snr se aumente α para reseterar el promedio anterior y luego ir bajando α hasta que se estabilice el promedio.

c) Analizar el archivo ejercicio_7_c.grc, y utilizar un archivo wav que tenga en su equipo. Aumentar la potencia de ruido hasta el punto en que se comienza a sentir ruido y registrar estos dbs. Modificar el archivo anterior para utilizar un codificador de Trellis y decodificador de Viterbi Combo. Volver a calcular ahora el punto en el que comienza a sentirse ruido y registrar los dbs. ¿Cuál es la ganancia en dbs en la calidad percibida del audio por utilizar un código de corrección de errores,

Bibliografía

[1] Cover, Thomas M. and Thomas, Joy A., Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing), 2006, isbn = 0471241954, Wiley-Interscience,

[2] Abramson N., Teoría de la Información y de la CODIFICACIÓN, Paraninfo, Madrid., 1986.

[3] Goldsmith, Andrea, Wireless Communications, 2005, 0521837162, Cambridge University Press, New York, NY, USA, Chapter 8

[4] Proakis, J. and Salehi, M., Digital Communications, isbn=9780072957167, 2007, McGraw-Hill Education

[5] Shu Lin and Daniel Costello, Error Control Coding, isbn=0-13-042672-5, 2004, Pearson Prentice Hall

[6] Curso: Digital Communications , MIT, Lectures: 7,8,9, http://web.mit.edu/6.02/www/f2010/