Temas Avanzados en Códigos para Corrección de Errores
Proyecto de implementación

Consideraciones generales.

- El trabajo consiste en la implementación, en software, de algoritmos de codificación y decodificación de códigos estudiados en el curso. Junto al desarrollo del software, se resolverán algunos problemas teóricos relacionados a los códigos y los algoritmos, y se realizarán experimentos que permitan por un lado verificar que el software funciona como se espera, y por otro entender el desempeño práctico de los códigos.

- Los programas deberán cumplir con los requisitos siguientes:

  - El lenguaje de programación debe ser C o C++.
  - Los experimentos solicitados en el trabajo deben ser compilables, ejecutables, y reproducibles por los docentes en plataformas de computación genéricas (PC con Windows o Linux, Mac, etc.), sin requerir paquetes especiales. La entrega debe incluir instrucciones claras de cómo compilar y ejecutar los programas (incluyendo archivos makefile, proyectos de MS Visual Studio, etc., apropiados a la plataforma).
  - Los programas deben ser eficientes y permitir la realización de los experimentos requeridos, procesando el número requerido de bloques de código en tiempo razonable (segundos, no horas). Una vez depurado el programa, y para realizar los experimentos, recuerde compilar con máxima optimización del tiempo: opción -O3 con gcc/g++en Linux/Mac, opción /Ot o versión Release con Visual en Windows. No utilice la versión Debug (u opción -g en Linux/Mac) en los experimentos.

- En la letra del proyecto se especifica la funcionalidad mínima requerida. Decisiones de diseño que no estén definidas explícitamente quedan a criterio de cada estudiante. Mejoras en funcionalidad, si son interesantes y efectivas, recibirán puntaje extra.

- El trabajo de evaluación debe ser concebido e implementado en su totalidad por el grupo de estudiantes evaluado.

  - **Está prohibido** bajar soluciones de cualquier tipo, totales o parciales, de internet, o copiarlas de cualquier otra fuente, incluyendo ejercicios similares en ediciones anteriores del curso.

  Tenga en cuenta que transgresiones a esta prohibición son relativamente fáciles de identificar (existen herramientas sofisticadas de dominio público que permiten detectar con gran confiabilidad la copia, con o sin alteraciones, de trabajos existentes, sea de internet o de otras fuentes).

- Se permite utilizar el resultado de un inciso teórico en un inciso posterior, aunque no se haya podido demostrar el resultado.

- El funcionamiento correcto de los programas es condición necesaria para la aprobación del proyecto.
- La entrega del proyecto se hará en forma electrónica, y consistirá en:

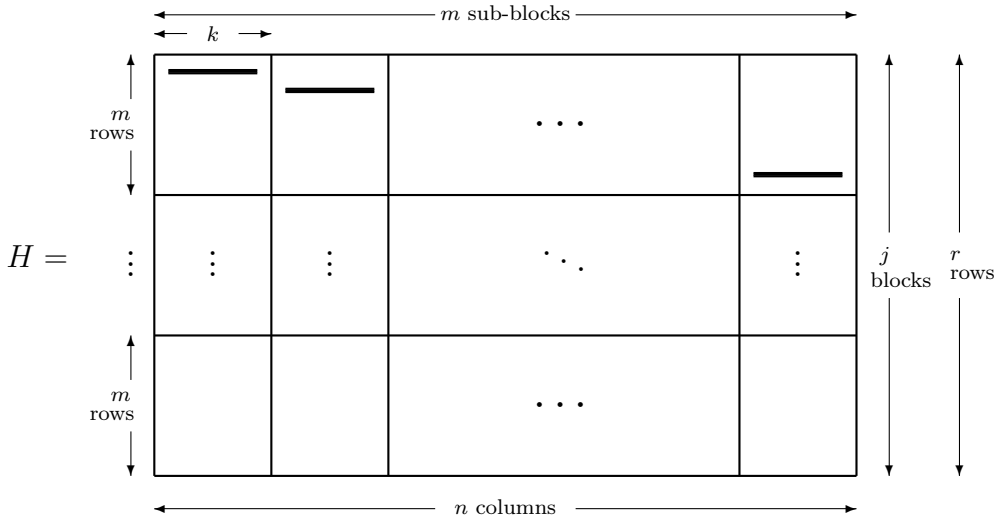
    1. Una copia del enunciado (este archivo).
    2. Los archivos con el código fuente de los programas, de manera que puedan ser compilados y ejecutados independientemente. Incluya instrucciones detalladas de compilación y ejecución, mencionando el sistema operativo y compilador con que se desarrolló el programa.
    3. Un informe conteniendo documentación del código entregado, los resultados de los experimentos, incluyendo tablas y gráficos, así como las respuestas y desarrollos correspondientes a las preguntas teóricas.
    4. Junto con los resultados, incluir en el documento copias exactas de las líneas de comando de los utilitarios empleados. En el caso de utilitarios que utilicen números seudo-aleatorios (p.ej. simuladores de canal), incluir en el informe las semillas utilizadas por los generadores. Todos los experimentos deben ser reproducibles.

Códigos LDPC y decodificación iterativa (pasaje de mensajes)

1. Introducción

Estudiaremos un código LDPC y su desempeño en el canal binario simétrico (BSC), utilizando un algoritmo de decodificación iterativo.

El código pertenece a la familia $\mathcal{G}(n, j, k)$ de Gallager, con parámetros que se especificarán abajo. Recordamos que la matriz de chequeo de paridad binaria H asociada con el código tiene la forma siguiente.



Las barras negras en el primer bloque horizontal de $m \times n$ representan filas de k unos. Los $j-1$ bloques horizontales de $m \times n$ que siguen se construyen tomando permutaciones aleatorias de las columnas de ese primer bloque. El entero m es arbitrario, y determina las dimensiones de la matriz, que contiene exactamente k unos por fila, y j unos por columna, $j < k$. Los parámetros del código satisfacen: $n = m \cdot k$, $r = m \cdot j$, y la tasa del código es $R \geq 1 - \frac{r}{n} = 1 - \frac{j}{k}$. Denominaremos a este código $\mathcal{C}_H$.

2. Tareas

2.1. Problema teórico

Suponemos que los parámetros j, k, m están dados en el contexto. Denotamos con 1^m un bloque de m unos, y, similarmente, con 0^m un bloque de m ceros. Sea ℓ un entero $0 \leq \ell < j$. Definimos el vector

$$\mathbf{u}(\ell) = [\underbrace{0^m\, 0^m \ldots 0^m}_{\ell}\; 1^m\; \underbrace{0^m \ldots 0^m}_{j-\ell-1}]$$

1. Verifique que $\mathbf{u}(\ell)$ tiene un largo total de r bits.

2. Demuestre que los vectores $\mathbf{u}(0) + \mathbf{u}(\ell)$, $0 < \ell < j$, son linealmente independientes y satisfacen
$$(\mathbf{u}(0) + \mathbf{u}(\ell))\, H = \mathbf{0}.$$
(La aritmética es en GF(2). Pista: demuestre que $\mathbf{u}(\ell)H$ no depende de ℓ.)

3. Sea r' el rango de H, que es igual a la redundancia de $\mathcal{C}_H$. Demuestre que $r' \leq r - j + 1$.

2.2. Implementación

Fijamos los parámetros $j = 3$, $k = 4$, $m = 400$, $n = 1600$, $r = 1200$, y elegimos una matriz H fija, que estará disponible en la página EVA del curso.

1. Implemente un decodificador iterativo para $\mathcal{C}_H$. El decodificador será del tipo "pasaje de mensajes" visto en clase, utilizando la Variante 1. Ver las diapositivas del curso por una descripción más detallada del algoritmo.

   Detalles de la implementación:

   a) Formato de la matriz H. La matriz H será provista en forma de un par de archivos `H.c`, `H.h` declarando un arreglo de 1200×1600 bytes en lenguage C. Cada byte tiene valor 0 o 1 y representa un bit de la matriz binaria. El archivo puede ser compilado tal cual como parte del programa del decodificador, o procesado de otra manera según criterio del grupo.

   b) El decodificador recibe como entrada un archivo de bloques codificados y posiblemente corruptos por el canal, y un argumento `iter` especificando el número máximo de iteraciones por bloque.

   A su vez, saca como salida un archivo de bloques decodificados de largo n, e imprime estadísticas de bloques decodificados correctamente y bloques fallidos (bloques en los que se llegó al número máximo de iteraciones sin producir una palabra de código, o bloques en que la palabra de salida falla los chequeos de paridad del código por algún motivo).

   c) Formato de los datos. Para facilitar la implementación se utilizará un formato algo ineficiente, donde cada bit de entrada o salida se representa con un byte de valor 0 o 1. O sea que un bloque de código ocupa, en los archivos de entrada y salida, 1600 bytes. Cada bloque $\mathbf{y}$ se lee (o escribe) en orden *ascendiente* de índice, $y_0, y_1, \ldots, y_{n-1}$, que se corresponde con el orden e indexado de las columnas de H.

2.3. Experimentos

1. Corra experimentos de corrupción en el canal y decodificación para los valores siguientes del parámetro p del BSC:
$$\{0{,}05, 0{,}06, 0{,}07, 0{,}08, 0{,}09, 0{,}100, 0{,}110, 0{,}120\}.$$

Para cada valor del parámetro, corra experimentos con `iter`=20 e `iter`=50. Utilice por lo menos 20000 bloques en cada experimento. Más detalles:

a) Para simplificar los experimentos y obviar la necesidad de emplear un codificador, supondremos que todas las palabras de código transmitidas son cero. *Esta suposición no debe ser utilizada en forma alguna por el decodificador, que debe ser general y funcionar correctamente con palabras de código arbitrarias.*

Para crear archivos de prueba donde datos de valor cero son corrompidos por el BSC, se utilizará un utilitario `bbchannel`, que estará disponible en la página EVA del curso. Por ejemplo, para generar 32000000 bits (originalmente de valor cero) afectados por un BSC de parámetro $p = 0{,}05$, y guardarlos en un archivo `testdata.dat` se utiliza la línea de comando

```
bbchannel -s 314 -p 0.05 -Z 32000000 testdata.dat
```

El argumento `-s 314` fija una semilla pseudo-aleatoria, que puede elegirse arbitrariamente. Documente las semillas utilizadas, para poder reproducir los experimentos.

Corra `bbchannel -h` para obtener más detalles de las opciones disponibles.

b) Utilice el utilitario `diffblock` para obtener una estimación empírica de P_{fail}, la probabilidad de falla de decodificación de bloque. Con la opción `-z`, el utilitario recibe solamente el archivo decodificado de entrada, y compara con bloques todos cero.

Tabule y grafique P_{fail} vs. p para los dos valores probados de `iter` (las dos gráficas en la misma figura). Discuta los resultados.

c) Compare la estimación de P_{fail} producida por `diffblock` con la producida por su programa en cada experimento (de acuerdo a lo pedido en el inciso *b*) de 2.2). Discuta los resultados.

2. Consideramos un código BCH binario $\mathcal{C}_{\text{bch}}$ de parámetros $n = 1600$, $k = 401$, $d = 219$. Cuando se decodifica con un algoritmo convencional que corrige hasta $(d-1)/2$ errores, se pueden estimar los valores siguientes de P_{fail} vs. p para este código.

p	P_{fail}
0,050	$6{,}22e{-}04$
0,060	$7{,}98e{-}02$
0,070	$5{,}92e{-}01$
0,080	$9{,}58e{-}01$
0,090	$9{,}99e{-}01$

- Verifique que $\mathcal{C}_{\text{bch}}$ tiene el mismo largo y *más redundancia* que $\mathcal{C}_H$ (pista: recuerde el problema teórico).
- Compare las estimaciones de P_{fail} para $\mathcal{C}_{\text{bch}}$ y sus estimaciones empíricas para $\mathcal{C}_H$ (utilice el valor de `iter` con los mejores resultados). Tabule, grafique, y discuta la comparación.