FARAUTE Ciens. y Tec., 2(2): 35-45, 2007 ISSN 1698-7418
Depósito Legal PP200402CA1617

ALGORITMOS DE CODIFICACIÓN Y DECODIFICACIÓN EFICIENTE UTILIZANDO CÓDIGOS HAMMING *

Efficient Coding and Decoding Algorithms Using Hamming Codes

DANIEL H. ROSQUETE DE M., AMADÍS A. MARTÍNEZ M. y FREDDY PEROZO R.
Universidad de Carabobo, Facultad de Ciencias y Tecnología, Dpto. de Computación.
{dhrosquete,aamartin,fperozo}@uc.edu.ve

Fecha de Recepción: 26/02/2007, **Fecha de Revisión:** 03/08/2007, **Fecha de Aceptación:** 31/10/2007

Resumen

En telecomunicaciones, el código Hamming es un código detector y corrector de errores que lleva el nombre de su inventor, Richard Hamming. Los códigos Hamming pueden detectar errores en uno o en dos *bits*, y también corregir errores en un solo *bit*. Éstos siguen siendo uno de los códigos correctores de errores más importantes desde diversos puntos de vista, tanto teóricos como prácticos. Han sido estudiados durante más de cuatro décadas y hasta ahora se han propuesto muchas propiedades relacionadas con ellos. En este artículo se presentan dos algoritmos, basados en el método original de Hamming, para los procesos de codificación y decodificación utilizando códigos Hamming. Se demuestra que la complejidad computacional de ambos algoritmos es menor que la de enfoques clásicos basados en álgebra lineal. Los algoritmos propuestos fueron probados sobre distintos conjuntos de datos. Los experimentos realizados confirman que los algoritmos desarrollados, en la práctica, son más rápidos.

Palabras clave: Codificación, Código Corrector de Errores, Código Hamming, Decodificación.

Abstract

In telecommunications, the Hamming code is an error detecting and correcting code named in honor to its creator, Richard Hamming. The Hamming codes can detect single and double-bit errors and correct single-bit errors as well. These still are one of the most important error correcting codes both from theoretical and practical points of view. These have been studied for more than four decades and many properties concerning them have been proposed so far. In this paper we present two algorithms, based on the original Hamming method, for the Hamming encoding and decoding processes. The computational complexity of both algorithms is proved to be smaller than the complexity of the classical approaches based on linear algebra. The proposed algorithms have been tested on different data sets. The experiments we made confirm that the developed algorithms are faster in practice too.

Keywords: Encoding, Error-Correcting Code, Hamming Code, Decoding.

*Una versión preliminar de este artículo fue presentada en la XXXII Conferencia Latinoamericana de Informática (CLEI 2006), realizada en Santiago de Chile, Chile, del 20 al 25 de agosto de 2006. Este artículo es una versión resumida de los resultados preliminares obtenidos en el Trabajo Especial de Grado titulado: "Algoritmos Eficientes para Códigos Hamming y Estudio de su Aplicación sobre Sistemas Existentes" (Rosquete De Mora, 2007)

1. Introducción

La comunicación es, básicamente, un proceso de emisión y recepción de mensajes. Los elementos principales de la comunicación son: el emisor, el mensaje, el receptor y el canal, donde el emisor envía un mensaje al receptor por medio de un canal. Un sistema de comunicaciones proporciona toda la infraestructura necesaria para que este proceso se lleve a cabo.

El acelerado desarrollo de los sistemas de comunicaciones ha incrementado el número de canales disponibles como, por ejemplo, líneas telefónicas, enlaces de radio, dispositivos de almacenamiento magnético u óptico, entre otros. Existen diversos factores (distorsión, interferencia, radiación, magnetización) que introducen ruido en la transmisión de datos sobre los canales de comunicación. Cuando existe ruido al transmitir datos sobre un canal, es probable que el mensaje recibido por el receptor no sea idéntico al mensaje enviado por el emisor. Cuando esto ocurre, se dice que se produjeron errores en la transmisión de datos sobre el canal.

Por esta razón se crea la teoría de la codificación, que estudia la transmisión de datos sobre canales de comunicación con ruido, y realiza la búsqueda de códigos para la detección y la corrección de errores introducidos en el canal. Sin embargo, es inútil tener algoritmos para codificar, decodificar, detectar y corregir errores si no son eficientes o no permiten elevadas velocidades de transmisión de datos.

Existen diversos métodos para abordar este problema, uno de los más conocidos es el código Hamming, publicado por Richard Hamming en 1950 (Hamming, 1950). Su propuesta, aún vigente, consiste en agregar redundancia a los datos, a través de *bits* de paridad (o *bits* de control) colocados en posiciones específicas, de manera tal que permitan detectar la presencia de errores dentro del mensaje, lo que proporciona al receptor la posibilidad de corregirlos. Los códigos Hamming pueden detectar errores en uno o en dos *bits*, y también corregir errores en un solo *bit*. Siguen siendo los códigos correctores de errores más importantes desde diversos puntos de vista, tanto teóricos como prácticos, en sistemas modernos de comunicaciones o almacenamiento digital.

Los códigos Hamming pueden implementarse tanto en *hardware* como en *software*. Los métodos por *hardware*, en general, tienden a ser más eficientes. Sin embargo, requieren un gran número de componentes lo que, además de los costos asociados, los hace inaplicables en algunos contextos como, por ejemplo, comunicaciones micro-satelitales (Saturno, 2003). Pese a que los métodos por *software* se consideran menos eficientes, esto es cierto para códigos sofisticados de detección de errores como, por ejemplo, *Cyclic Redundancy Check* (CRC). Existen códigos alternativos a CRC (entre ellos los de Hamming) que, además de proporcionar la posibilidad de realizar implementaciones eficientes por *software* (Feldmeier, 1995; Nguyen, 2005), ofrecen varias ventajas: independencia de la plataforma, simplicidad de implementación y manipulación más conveniente de la información (*Bytes* o palabras en lugar de *bits*). Estas ventajas y la poca disponibilidad de trabajos que no utilicen enfoques basados en álgebra lineal (Jacobsmeyer, 2004), motivaron la búsqueda de algoritmos eficientes no tradicionales para la codificación y decodificación utilizando códigos Hamming.

En este artículo se presentan dos algoritmos, basados en el método original de Hamming (Hamming, 1950), para los procesos de codificación y decodificación de códigos Hamming. Posteriormente, se de-

muestra que la complejidad computacional de ambos algoritmos es menor que la de enfoques clásicos basados en álgebra lineal. Los algoritmos propuestos fueron probados sobre distintos conjuntos de datos; los experimentos realizados confirman que los algoritmos desarrollados, en la práctica, son más rápidos.

Este artículo fue estructurado en ocho secciones, incluyendo la introducción. En la sección 2 se introducen conceptos relacionados con el problema de transmisión de datos y teoría de la codificación. La sección 3 presenta resultados teóricos relacionados con la complejidad computacional de las tareas algorítmicas involucradas en el problema de transmisión de datos. En la sección 4 se describe brevemente el método original de Hamming. En la sección 5 se presentan los algoritmos propuestos para la codificación y decodificación utilizando códigos Hamming. La sección 6 contiene el análisis de complejidad computacional y optimalidad de los algoritmos propuestos. En la sección 7 se reportan los resultados experimentales obtenidos por los algoritmos desarrollados. Finalmente, la sección 8 contiene las conclusiones del trabajo y las recomendaciones finales.

2. Definiciones previas [1]

Un sistema digital de comunicaciones puede modelarse utilizando un diagrama de bloques, como se presenta en la Fig. 1 (Jacobsmeyer, 2004).

Basado en este modelo, los mensajes que se envían desde el emisor son una secuencia de símbolos tomados de un alfabeto S. Dado que los sistemas digitales de comunicaciones utilizan equipos que manejan datos binarios, se asume que el alfabeto a utilizar contiene sólo dos símbolos $S = \{0, 1\}$, cada uno de los cuales se denomina *bit*.

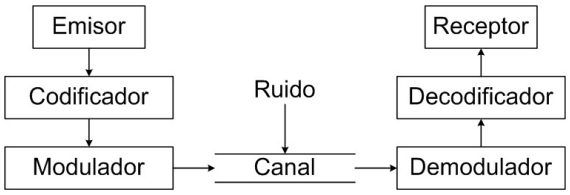

Fig. 1. Modelo de un sistema digital de comunicaciones

Para transmitir un mensaje, el emisor requiere de un canal de comunicaciones. Un canal es un dispositivo que recibe una secuencia de símbolos (*bits* en este caso) y los transporta en algún formato físico adecuado hasta el otro extremo del canal. Se dice que un canal es *noiseless* (silencioso) si la secuencia de *bits* enviada por el emisor y la recibida por el receptor son siempre iguales. En caso contrario, se dice que el canal es *noisy* (ruidoso).

Siguiendo el diagrama de la Fig. 1, la transferencia de datos desde el emisor hasta el extremo inicial del canal requiere de dos dispositivos: (1) el codificador, que modifica la secuencia de símbolos enviados por el emisor para crear un *codeword* (palabra de código) y (2) el modulador, que recibe la palabra de código generada por el codificador y la transforma a un formato físico adecuado para su transmisión por el canal. Una vez realizada la transmisión, en el extremo final del canal se encuentra el demodulador, que recupera la señal recibida por el canal y la transforma a formato digital. La secuencia de *bits* resultante de este proceso es recibida por el decodificador el cual, en caso de errores en la transmisión, intenta realizar la recuperación del mensaje original para entregarlo finalmente al receptor.

Formalmente, sea $S = \{0, 1\}$ un al-

[1] Todas las definiciones y conceptos introducidos en esta sección están basados en los presentados en (Hoffman et al., 1991; Jacobsmeyer, 2004; Mackay, 2005; Xambo-Descamps, 2004)

fabeto binario, cuyos elementos se denominan *bits*. Un código C de tamaño M y longitud N sobre el alfabeto S es un conjunto $C = \{c_1, c_2, \ldots, c_M\} \subseteq S^N$. Los elementos $c_k \in C$ $(1 \leq k \leq M)$ se denominan *codewords* (palabras de código), M define el tamaño del código C (el número de palabras de código contenidas en C) y cada palabra de código es una secuencia de longitud N *bits*. Los elementos de S^N pueden escribirse, de manera equivalente, utilizando notación de N-tuplas de la forma $x = (x_1, x_2, \ldots, x_N)$ o concatenación de símbolos como $x = x_1 x_2 \ldots x_N$, donde $\langle \forall i : 1 \leq i \leq N : x_i \in \{0, 1\} \rangle$. Un codificador es una función biyectiva $f : S^n \rightarrow C$, donde $n < N$ es un entero positivo y $C \subseteq S^N$. $C = f(S)$ es el conjunto de palabras de código del codificador. De manera análoga, un decodificador es una función $g : C \rightarrow S^n$ tal que $\langle \forall x : x \in C : g(f(x)) = x \rangle$. Sea m el mensaje enviado por el emisor, $t = f(m)$ el mensaje codificado y t' el mensaje recibido. Cuando t' llega al decodificador, se presentan dos posibilidades: (1) $t' \in C$, en este caso se decodifica $m' = g(t')$ que probablemente coincidirá con m o (2) $t' \notin C$, en este caso se detecta un error. El detector tiene tres alternativas: (1) descartar el mensaje recibido, (2) solicitar al emisor la retransmisión del mismo o (3) corregir el error si tiene la lógica adicional necesaria.

3. Resultados teóricos

En esta sección se incluyen resultados teóricos relacionados con la complejidad computacional de las tareas algorítmicas involucradas en el problema de transmisión de datos.

La primera tarea corresponde con la resolución algorítmica del problema de codificación. Como se explicó en la sección 2, la transferencia desde el emisor hasta el extremo inicial del canal requiere de un codificador, el cual modifica la secuencia de *bits* enviados por el emisor para crear una palabra de código. En este contexto, el problema de codificación puede plantearse como sigue: Dada una secuencia de n símbolos $m = x_1 x_2 \ldots x_n$, donde $\langle \forall i : 1 \leq i \leq n : x_i \in \{0, 1\} \rangle$, transformarla en una palabra de código de N símbolos $f(m) = y_1 y_2 \ldots y_N$, donde $n < N$ y $\langle \forall j : 1 \leq j \leq N : y_j \in \{0, 1\} \rangle$. Este problema, utilizando enfoques clásicos basados en álgebra lineal, tiene una complejidad en tiempo proporcional a $\mathcal{O}(Nn)$ (Sudan, 1997).

La segunda tarea corresponde con la resolución algorítmica del problema de decodificación. Para esta tarea, el enfoque se concentra en si la secuencia recibida de símbolos contiene errores o no. Para ello, se define el problema de detección de errores: Dada una secuencia recibida de N símbolos $R = r_1 r_2 \ldots r_N$, donde $\langle \forall j : 1 \leq j \leq N : r_j \in \{0, 1\} \rangle$, determinar si R es una palabra de código o no. Este problema, utilizando enfoques clásicos basados en álgebra lineal, también tiene una complejidad en tiempo proporcional a $\mathcal{O}(Nn)$ (Sudan, 1997).

Sin embargo, en un trabajo posterior (Ashikhmin & Litsyn, 2004) se demostró que, utilizando una modificación de la transformada de Walsh-Hadamard, es posible construir algoritmos para los problemas de codificación / decodificación con complejidad en tiempo proporcional a $\mathcal{O}(N \lg(n))$, donde n es el número de *bits* enviados por el emisor y N es el número de *bits* de la palabra de código (Ashikhmin & Litsyn, 2004; Djordjevic et al., 2005). En este artículo se presentan dos algoritmos para los procesos de codificación y decodificación de códigos Hamming cuya complejidad en tiempo es óptima con respecto a esta cota teórica de $\mathcal{O}(N \lg(n))$. Los algoritmos propuestos están basados en el

método original de Hamming (Hamming, 1950), y no utilizan los enfoques clásicos basados en álgebra lineal.

4. Descripción del método

En esta sección se describe brevemente el método original de Hamming (Hamming, 1950), el cual se puede dividir en dos etapas:

4.1. Codificación

En esta etapa se asignan los *bits* de paridad necesarios para controlar todos los *bits* de datos del mensaje. El número de *bits* de paridad requeridos para n *bits* de datos está determinado por la desigualdad: $2^p \geq n + p + 1$, donde p es el número de *bits* de paridad y n es el número de *bits* de datos. Se debe tener en cuenta que, en la palabra de código a generar, las posiciones que son potencia de dos $(1, 2, 4, 8, 16, 32, 64, \ldots)$ se utilizan como *bits* de paridad, mientras que el resto de las posiciones $(3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, \ldots)$ son utilizadas como *bits* de datos. La posición de cada *bit* de paridad en la palabra de código que se genera en la etapa de codificación, determina la secuencia de los *bits* de datos que verifica en la misma, tal como se presenta en la Tabla 4.1.

El valor de cada *bit* de paridad se obtiene sumando la cantidad de unos que hubo en las posiciones comprobadas; si esta suma es impar, el valor del *bit* de paridad evaluado es 1, en caso contrario el valor es 0.

Por ejemplo, suponga que se desea enviar la secuencia `0101`. En la nueva palabra de código se asignarían los *bits* de datos de la siguiente forma: `_ _ 0 _ 1 0 1`, donde el símbolo `_` (*underscore*) representa las posiciones en las que se asignarán los *bits* de paridad. Siguiendo el método de Hamming, se obtienen los valores presentados en la Tabla 4.1.

Finalmente, la palabra de código (mensaje codificado) que se genera es la siguiente: `0 1 0 0 1 0 1`.

4.2. Decodificación

La etapa de decodificación en el receptor tiene dos fases, la primera obligatoria (detección) y la segunda opcional (corrección).

4.2.1. Código de detección de errores en un solo *bit*

Esta fase detecta cuando existe un solo *bit* dañado: una vez recibido el mensaje, se genera una nueva palabra de código recalculando sobre las posiciones de los *bits* de paridad (sección 4.1), y se compara con la secuencia recibida. Si son iguales, significa que no hubo errores; en caso contrario, se detecta la existencia de un error y se procede a utilizar el código de corrección de errores.

4.2.2. Código de corrección de errores en un solo *bit*

En esta fase, se compara la secuencia recibida con la palabra de código calculada en la etapa anterior, y se suman las posiciones de los *bits* de paridad que difieran entre ellas. El resultado de esta suma determina la posición del *bit* dañado, se sustituye por el valor opuesto y concluye esta fase.

Por ejemplo, suponga que se desea enviar desde el emisor la palabra de código `0 1 0 0 1 0 1`, en la transmisión ocurre una falla y al receptor llega la secuencia `0 1 0 0 1 0 0`, se debe destacar que el *bit* dañado es el último de esta secuencia. Al realizar el nuevo cálculo sobre la palabra de

Bit de Paridad	Posición	Posiciones que comprueba
1	1	$1, 3, 5, 7, 9, 11, 13, 15, 17, \ldots$
2	2	$2, 3, 6, 7, 10, 11, 14, 15, 18, \ldots$
3	4	$4, 5, 6, 7, 12, 13, 14, 15, 20, \ldots$
4	8	$8, 9, 10, 11, 12, 13, 14, 15, 24, \ldots$

Tabla 1. Relación entre *bits* de paridad y *bits* de datos

Bit de paridad	Posición	Posiciones comprobadas	Datos evaluados	Suma	Valor
1	1	$1, 3, 5, 7$	_, 0, 1, 1	2	0
2	2	$2, 3, 6, 7$	_, 0, 0, 1	1	1
3	4	$4, 5, 6, 7$	_, 1, 0, 1	2	0

Tabla 2. Corrida de la codificación

código _ _ 0 _ 1 0 0, se obtienen los valores presentados en la Tabla 4.2.2. Luego, se compara la secuencia recibida (0 1 0 0 1 0 0) con la nueva palabra de código (1 0 0 1 1 0 0). Como se puede observar, los valores de los *bits* de paridad 1, 2 y 3 no coinciden. Esto implica que, al sumar las posiciones que ocupan estos *bits* de paridad, $1 + 2 + 4 = 7$, se obtiene la posición del *bit* dañado, el cual deberá reemplazarse por el valor opuesto (1 en este caso) para realizar la corrección.

5. Descripción del algoritmo propuesto

En esta sección se presentan los algoritmos, basados en el método original de Hamming (Hamming, 1950), para los procesos de codificación y decodificación utilizando códigos Hamming. En la subsección 5.1 se explican brevemente el codificador y el decodificador. La subsección 5.2 contiene los algoritmos propuestos, junto con una explicación de su funcionamiento, siguiendo el formato y los lineamientos presentados en (Cormen et al., 2001).

5.1. Codificador y decodificador

El codificador propuesto está constituido por dos componentes secuenciales: el preprocesador y el *encoder*. El preprocesador es el componente encargado de calcular el número de *bits* de paridad requeridos, basándose en la regla de Hamming (Hamming, 1950), copiar los n *bits* de datos del mensaje a la palabra de código de longitud N e inicializar el valor de los *bits* de paridad. El *encoder* es el componente encargado de determinar el valor de los *bits* de paridad, basándose en el método original de Hamming (Hamming, 1950), descrito en la sección 4.

El decodificador propuesto está constituido por dos componentes secuenciales: el detector y el corrector. El detector es el algoritmo encargado de la detección de errores en uno o en dos *bits*: si hay dos *bits* dañados, se solicita la retransmisión del mensaje; si hay un solo *bit* dañado, se pasa a la fase de corrección. El corrector es el algoritmo encargado de la corrección de errores en un solo *bit*. El funcionamiento de ambos también está basado en el méto-

Bit de paridad	Posición	Posiciones comprobadas	Datos evaluados	Suma	Valor
1	1	1, 3, 5, 7	_, 0, 1, 0	1	1
2	2	2, 3, 6, 7	_, 0, 0, 0	0	0
3	4	4, 5, 6, 7	_, 1, 0, 0	1	1

Tabla 3. Codificación de los datos erróneos

do original de Hamming (Hamming, 1950), descrito en la sección 4.

5.2. Algoritmos propuestos

El Algoritmo 1 presenta el proceso de codificación. El algoritmo para la decodificación (Algoritmo 3) es similar al de codificación en lo que respecta a la generación de una nueva palabra de código para su posterior comparación con la secuencia recibida, incluyendo luego las acciones necesarias para detectar errores en uno o en dos *bits*, y también corregir errores en un solo *bit*.

Las variables globales requeridas por estos algoritmos son las siguientes:

- *code*[1..*N*]: contendrá la palabra de código que se procesará a lo largo del método Hamming.
- *paridad*: almacena la cantidad de *bits* de paridad requeridos.
- *size*: contiene la longitud de la palabra de código.
- *tot*: contabiliza el número de unos asociados con una determinada posición.

El procedimiento ENCODER (Algoritmo 1) requiere de una fase previa (preprocesador) en la que se determina el valor de la variable *paridad*, se copian los *bits* de datos del mensaje a la palabra de código *code*[1..*N*] y se inicializan los *bits* de paridad con el valor 2. La variable *tot* se inicializa con el valor 0 (línea 1), para luego realizar un ciclo (líneas 2 – 9) cuyo número de iteraciones está determinado por el valor de la variable *paridad*. Antes de cada iteración del ciclo interno (línea 3), se asigna el valor de 2^i a la variable local L, para almacenar la cantidad de *bits* que se van a revisar en esa iteración. A la variable local j se le asigna el valor de L, para que el ciclo comience desde la posición del *bit* de paridad a evaluar. El ciclo interno (líneas 4 – 6) comprueba todas las posiciones correspondientes con la posición del *bit* de paridad que se está evaluando. Por ejemplo, si se está analizando el *bit* 2 de paridad, el algoritmo sólo verificaría las posiciones $2-3$, $6-7$, $10-11$, y así sucesivamente (Tabla 4.1). Esto se logra utilizando el procedimiento CHECK (Algoritmo 2), que recibe como parámetros los valores de L y de j. Una vez en este procedimiento, se asigna a la variable *cota* la suma $L+j-1$ (línea 1), esta fórmula determina hasta cuál posición se debe verificar. Luego se valida que el valor *cota* no exceda el número máximo de posiciones (*size*); de ser así, se corrige *cota* con el valor máximo de posiciones para evitar iteraciones adicionales (línea 2). El ciclo que sigue a esta validación (líneas 3 y 4) revisa las posiciones desde j hasta *cota*, asegurando que la suma almacenada en *tot* no contará el *bit* de paridad que se está evaluando. Al salir del procedimiento CHECK, se realiza una asignación al índice j, con la finalidad de que revise sólo las posiciones relevantes para el *bit* de paridad evaluado. De esta manera se realizan todas

```
    ENCODER()
(1)  tot ← 0;
(2)  for i ← 0 to paridad − 1
(3)     L ← 2^i; j ← L;
(4)     while (j ≤ size)
(5)        CHECK(L, j);
(6)        j ← j + 2L;
(7)     if (tot mod 2 = 0) then code[L] ← 0
(8)                        else code[L] ← 1
(9)     tot ← 0;
```

Algoritmo 1: Proceso de codificación

```
    CHECK(L, j)
(1)  cota ← L + j − 1;
(2)  if (cota > size) then cota ← size
(3)  for k ← j to cota
(4)     if (code[k] ≠ 2) then tot ← code[k] + tot
```

Algoritmo 2: Recorrido de las posiciones a verificar

```
    DECODER()
(1)  tot ← 0;
(2)  for i ← 0 to paridad − 1
(3)     L ← 2^i; j ← L;
(4)     while (j ≤ size)
(5)        CHECK(L, j);
(6)        j ← j + 2L;
(7)     if (tot mod 2 = 0) then code[L] ← 0
(8)                        else code[L] ← 1
(9)     tot ← 0;
(10) sum ← 0;
(11) damaged ← 0;
(12) for i ← 0 to paridad − 1
(13)    L ← 2^i;
(14)    if (code[L] ≠ copia[L]) then sum ← sum + L; damaged ← 1;
(15) if ((damaged = 1) ∧ (sum ≤ size)) then code[sum] ← ¬code[sum]
```

Algoritmo 3: Proceso de decodificación

las iteraciones, hasta que j es mayor que $size$. Finalmente, se comprueba si el valor final de tot es par, de ser así se asigna 0 en la posición del *bit* de paridad evaluado; en caso contrario, se asigna 1 (líneas 7 y 8). Luego se re-inicializa la variable tot (línea 9), para repetir este proceso tantas veces como *bits* de paridad se requieran.

Las líneas 1 – 9 del procedimiento DECODER (Algoritmo 3) tienen un funcionamiento análogo al explicado para ENCODER, para generar así una nueva palabra de código recalculando sobre las posiciones de los *bits* de paridad. Las líneas 12 – 14 realizan la comparación de la secuencia recibida con la nueva palabra de código generada, para la detección de errores en uno o en dos *bits*. Finalmente, la línea 15 se encarga de la corrección de errores en un solo *bit*, la cual sólo se realiza en caso de detección

positiva.

Se debe mencionar que los códigos Hamming son capaces de detectar si ocurrió más de 1 error en la transmisión, siendo necesarias las siguientes condiciones: (1) el mensaje debe ser mayor que 7 *bits* en total, es decir, que existan por lo menos 4 *bits* de paridad y (2) debe existir máximo 1 error por cada bloque cubierto por *bit* de paridad, por ejemplo, el *bit* 3 de paridad controlará de la posición 9 a la 15, donde sólo debe existir a lo sumo 1 error.

6. Análisis de complejidad y optimalidad

El número de iteraciones del ciclo externo del codificador (Algoritmo 1) está determinado por el número de *bits* de paridad requeridos para n *bits* de datos. Este número se calcula previamente, basándose en la regla de Hamming (Hamming, 1950): $2^p \geq n+p+1$, donde n es el número de *bits* de datos y p es el número de *bits* de paridad. De aquí se obtiene que $p \geq \lg(n+p+1)$ y el número de *bits* de paridad tiende a $\lceil \lg(n) \rceil$ a medida que aumenta el valor de n. En el ciclo interno, se determina el valor de cada uno de los *bits* de paridad. De la sección 4 se conoce que cada *bit* de paridad se obtiene calculando la paridad de alguno de los *bits* de datos, es decir, la posición del *bit* de paridad determina la secuencia de los *bits* de datos que comprueba. En cada una de estas iteraciones, se consideran a lo sumo $\lceil N/2 \rceil$ *bits* de datos. Aplicando las reglas del análisis asintótico (Aho et al., 1983; Cormen et al., 2001), se tiene que el número máximo de iteraciones es a lo sumo $\lceil N/2 \rceil \lceil \lg(n) \rceil$. Esto significa que la complejidad en tiempo del algoritmo de codificación es $T(n) = \mathcal{O}(N \lg(n))$.

El uso de memoria del algoritmo de codificación está determinado por el vector $code[1..N]$, en el que se almacena la secuencia que se procesará a lo largo del método Hamming. Esto significa que la complejidad en espacio del codificador es $S(n) = \mathcal{O}(N)$.

El análisis del decodificador (Algoritmo 3) es similar al del codificador en lo que respecta a la generación de una nueva palabra de código para su posterior comparación con la secuencia recibida, por lo que la complejidad en tiempo de esta tarea es $T_1(n) = \mathcal{O}(N \lg(n))$. Las acciones necesarias para detectar errores en uno o en dos *bits*, y también corregir errores en un solo *bit*, involucran un recorrido sobre la palabra de código, por lo que la complejidad en tiempo de esta tarea es $T_2(n) = \mathcal{O}(N)$. Aplicando las reglas del análisis asintótico (Aho et al., 1983; Cormen et al., 2001), se tiene que la complejidad en tiempo del decodificador viene dada por $T(n) = \mathcal{O}(\text{máx}(N \lg(n), N)) = \mathcal{O}(Nlg(n))$.

El uso de memoria del algoritmo de decodificación está determinado por los vectores $code[1..N]$ y $copia[1..N]$, lo que significa que la complejidad en espacio del decodificador es $S(n) = \mathcal{O}(N)$.

En la sección 3 se presentaron resultados teóricos relacionados con la complejidad computacional de las tareas algorítmicas de codificación y decodificación. En ambos casos la complejidad en tiempo, utilizando enfoques clásicos basados en álgebra lineal, es proporcional a $\mathcal{O}(Nn)$ (Sudan, 1997). También se presentó la posibilidad de construir algoritmos para la resolución de los problemas de codificación / decodificación con complejidad en tiempo proporcional a $\mathcal{O}(N \lg(n))$ (Ashikhmin & Litsyn, 2004; Djordjevic et al., 2005). Por lo tanto, una vez determinada la complejidad en tiempo de los algoritmos propuestos, se puede concluir que son óptimos con respecto a las cotas presentadas.

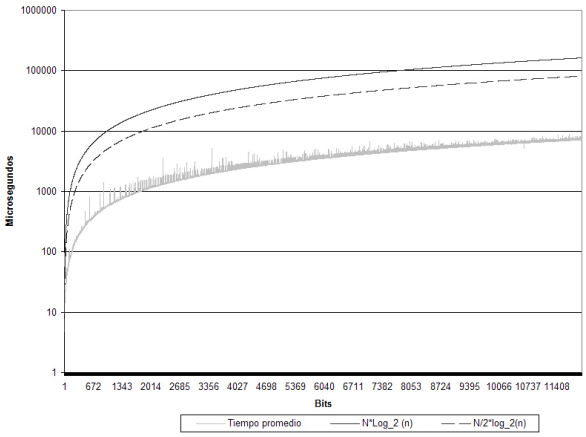

Fig. 2. Desempeño promedio de los algoritmos implementados

7. Resultados experimentales

La implementación de los algoritmos se realizó utilizando el lenguaje de programación ANSI C/C++, y las pruebas se realizaron en un computador con procesador *Intel Pentium* IV de 1.80 GHz y 512 MB de memoria RAM, bajo el sistema operativo *White Box Linux*.

Para la configuración de los experimentos, se realizaron diversos casos de prueba en los que se varió de manera ascendente (en un *bit*) la longitud del mensaje, desde 3 *bits* hasta 12.000 *bits* aleatorios (1500 *Bytes*), la unidad máxima de transmisión (*Maximum Transmission Unit* - MTU) para redes *Ethernet* (Postel, 1983; Hornig, 1984). Cada uno de estos casos de prueba fue codificado, alterado en 1 *bit*, decodificado y reparado por la implementación de los algoritmos propuestos. Debido a las prioridades de uso del procesador, cada caso se probó 3 veces y se calculó un tiempo promedio. Bajo este enfoque, se procesaron en total 76.380.694 *bits* y el tiempo total de procesamiento fue de 42 segundos. Los resultados individuales promedios se muestran en la Fig. 2, junto con las cotas establecidas en la sección 6.

El eje de las abscisas contiene la cantidad de *bits* evaluados (incluyendo los *bits* de paridad agregados posteriormente), mientras que el eje de las ordenadas denota el tiempo de procesamiento (en microsegundos). Se incluyen como referencia las funciones de cota $N \lg(n)$ (línea delgada continua) y $(N/2) \lg(n)$ (línea punteada). Se debe destacar que el desempeño promedio de los casos de prueba no supera las funciones establecidas de cota, lo que confirma experimentalmente la complejidad en tiempo presentada en la sección 6.

8. Conclusiones y trabajo futuro

En este artículo se presentaron dos algoritmos, basados en el método original de Hamming (Hamming, 1950), para los procesos de codificación y decodificación utilizando códigos Hamming. Se demostró formalmente que la complejidad computacional de ambos algoritmos es menor que la de enfoques clásicos basados en álgebra lineal. El estudio experimental realizado mostró que, en todos los casos de prueba, los algoritmos encuentran soluciones óptimas en tiempo con respecto a las funciones teóricas de cota presentadas en la sección 3. De esta manera, se puede destacar que efectivamente se logra un rendimiento óptimo para el procesamiento de códigos Hamming bajo el esquema propuesto, pasando por todas sus fases internas. Como trabajo futuro, se estudiará el rendimiento del esquema presentado en este trabajo desde varias perspectivas: (1) implementación por *hardware* de los algoritmos, dado que éste tiende a ser más eficiente, aunque no siempre aplicable (Saturno, 2003), (2) incorporación a protocolos de redes, para tratar de reducir el tiempo de procesamiento del *software* de comunicaciones y (3) planteamiento de diversas aplicaciones donde los algoritmos propuestos puedan o deban ser

utilizados, con sus respectivos estudios de viabilidad.

Referencias

Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). *Data Structures and Algorithms*. Massachusetts, US: Addison Wesley.

Ashikhmin, A. E., & Litsyn, S. (2004, August). Simple MAP Decoding of First-Order Reed-Muller and Hamming Codes. *IEEE Transactions on Information Theory*, *50*(8), 1812-1818.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms* (Second ed.). Massachusetts, US: The MIT Press.

Djordjevic, I. B., Milenkovic, O., & Vasic, B. (2005, May). Generalized Low-Density Parity-Check Codes for Optical Communication Systems. *Journal of Lightwave Technology*, *23*(5), 1939-1946.

Feldmeier, D. C. (1995, December). Fast Software Implementation of Error Detection Codes. *IEEE/ACM Transactions on Networking*, *3*(6), 640-651.

Hamming, R. W. (1950, April). Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, *26*(2), 147-160.

Hoffman, D. G., Wal, Leonard, D. A., Lidner, C. C., Phelps, K. T., & Rodger, C. A. (1991). *Coding Theory: The Essentials*. New York, NY, USA: Marcel Dekker, Inc.

Hornig, C. (1984, April). *A Standard for the Transmission of IP Datagrams over Ethernet Networks* (No. RFC894). Internet Engineering Task Force.

Jacobsmeyer, J. M. (2004). *Introduction to Error-Control Coding* (Tech. Rep.). Pericle Communications Company. (URL: `http://www.pericle.com/papers/Error_Control_Tutorial.pdf`)

Mackay, D. J. (2005). *Information Theory, Inference, and Learning Algorithms* (Fourth ed.). Cambridge, UK: Cambridge University Press.

Nguyen, G. D. (2005, January). Error-Detection Codes: Algorithms and Fast Implementation. *IEEE Transactions on Computers*, *54*(1), 1-11.

Postel, J. (1983, November). *The TCP Maximum Segment Size and Related Topics* (No. RFC879). Internet Engineering Task Force.

Rosquete De Mora, D. H. (2007, Abril). *Algoritmos Eficientes para Códigos Hamming y Estudio de su Aplicación sobre Sistemas Existentes*. (Trabajo Especial de Grado, dirigido por A. Martínez y F. Perozo. Universidad de Carabobo, Venezuela.)

Saturno, M. E. (2003). Software Error Protection Technic for High Density Memory. In *4th International Academy of Astronautics Symposium on Small Satellites for Earth Observation* (p. 406-409). Walter de Gruyter, Berlin, DE.

Sudan, M. (1997). Algorithmic Issues in Coding Theory. In S. Ramesh & G. Sivakumar (Eds.), *Foundations of Software Technology and Theoretical Computer Science* (Vol. LNCS1346, p. 184-199). Springer-Verlag.

Xambo-Descamps, S. (2004). *Block Error-Correcting Codes: A Computational Primer*. New York, US: Springer-Verlag.