IMPLEMENTACION DE UNA MODIFICACION DEL ALGORITMO DE HUFFMAN UTILIZANDO REDES NEURONALES

ING. WILLIAM JAVIER CASTILLO GÁMEZ

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

MAESTRÍA EN TELECOMUNICACIONES MÓVILES

SISTEMAS DE COMUNICACIONES DIGITALES E INTELIGENCIA COMPUTACIONAL

BOGOTÁ, COLOMBIA

2019

IMPLEMENTACION DE UNA MODIFICACION DEL ALGORITMO DE HUFFMAN UTILIZANDO REDES NEURONALES

ING. WILLIAM JAVIER CASTILLO GÁMEZ

TESIS DE GRADO PARA OPTAR POR EL TÍTULO DE MÁSTER EN TELECOMUNICACIONES

DIRECTORA:

ING. MARTHA RUTH OSPINA

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

MAESTRÍA EN TELECOMUNICACIONES MÓVILES

SISTEMAS DE COMUNICACIONES DIGITALES E INTELIGENCIA COMPUTACIONAL

BOGOTÁ, COLOMBIA

2019

CONTENIDO

CONTENIDO 3
INDICE DE GRÁFICAS 6
INDICE DE TABLAS 8
Resumen 9
Palabras Clave 9
Introducción 10
1. Problema de investigación 12
2. MARCO TEORICO Y REFERENCIAL 13
2.1 Marco Teórico 13
2.1.1 Sistemas de comunicaciones digitales 13
2.1.1.2 Codificación de la Fuente 14
2.1.1.3 Compresión de la Información 15
2.1.1.4 Compresión sin Pérdidas 16
2.1.1.5 Técnicas de Codificación y compresión de datos 16
2.1.1.5.1 Métodos Estadísticos 17
2.1.1.5.2 Métodos con uso de Diccionario de prefijos 17
2.1.2 Algoritmos de Codificación y compresión 17
2.1.2.1 Codificación Huffman 19
2.1.2.2 Codificación Aritmética 26
2.1.3 Complejidad Algorítmica 26
2.1.3.1 Complejidad Temporal 27
2.1.3.1.1 Cotas De Complejidad 30

2.1.3.2 Complejidad Espacial 32
2.1.4 Redes neuronales artificiales 32
2.1.4.1 Elementos de las RNA 33
2.1.4.2 Funciones de Activación 34
2.1.4.3 RNA Multicapa 35
2.1.4.4 RNA Feed Forward Back Propagation (FFBP) 36
2.1.4.4.1 Estructura de una red FFBP 36
2.1.4.4.2 Aprendizaje Backpropagation 37
2.1.4.4.3 Algoritmo Backpropagation 38
2.2 marco referencial 39
2.2.1 Sistemas de comunicaciones digitales, Codificación de la Fuente 39
2.2.2 Complejidad Algorítmica 40
2.2.3 Codificación con el método de Huffman 41
2.2.4 Redes Neuronales Artificiales 42
2.2.5 Algoritmo de Huffman y las aplicaciones con redes neuronales u otros métodos de Inteligencia computacional 43
3. desarrollo de la investigacion 45
3.1 propuesta del modelamiento matematico del algoritmo de huffman modificado 45
3.1.1 Modelado del Grafo de Huffman 45
3.1.1.1 Elementos matemáticos del Modelo 46
3.1.1.2 Conjunto A del grafo de Huffman 48
3.2 Diseño de modelos para el algoritmo 50
3.2.1 Modelo Estructural 50
3.2.1.1 Patrones de Diseño 51
3.2.1.2 Patrón Estructural Facade 51
3.2.1.3 Patrón Creacional Abstract Factory 52

3.2.1.4 Patrón estructural Bridge 54
3.2.1.5 Estructura general del Algoritmo 55
3.2.2 Diseño dinámico de comportamiento 57
3.2.2.1 Componente Estadístico 59
3.2.2.2 Subprocesos Estructurales del grafo Huffman 64
3.2.2.2.1 Cálculo de niveles del grafo 65
3.2.2.2.2 Proceso de Construcción del Grafo 66
3.2.2.2.3 Proceso de Codificación 71
3.2.2.3 Complejidad Computacional Total 72
3.2.3 Diseño de la Red Neuronal FFBP 74
3.2.3.1 Diseño Matemático de la RNA FFBP 75
3.3 implementacion 83
3.4 pruebas y analisis de resultados 85
3.4.1.1 Características de la fuente de información 88
3.4.2 Procedimiento de las pruebas 90
3.4.3 Resultados de las Pruebas 91
3.4.3.1 Cálculo Teórico de la Tasa de compresión 91
3.4.3.2 Resultados y análisis por Video 93
3.4.3.3 Resultados y análisis por Iteración 106
3.4.3.4 Resultados y análisis por tasa de compresión 109
3.4.3.5 Resultados y análisis de complejidad computacional 112
3.4.3.6 Análisis Integrado de los resultados 116
4. Aportes y trabajos futuros 120
5. conclusiones 123
6. REFERENCIAS 125

INDICE DE GRÁFICAS

Gráfica 1 Diagrama de Bloques Sistema de Comunicaciones Digitales 14
Grafica 2 Cronología de los Algoritmos de Compresión más Representativos 18
Gráfica 3 Estructura del árbol de Huffman nivel con mayor profundidad 21
Gráfica 4 Estructura del árbol de Huffman 2° nivel con número par de elementos 22
Gráfica 5 Estructura del árbol de Huffman 2° nivel con número impar de elementos 22
Gráfica 6 Estructura general del árbol de Huffman 23
Grafica 7 *Estructura general del árbol de Huffman con código en bits* 24
Grafica 8 Función Sigmoide Logarítmica 34
Grafica 9 Función Sigmoide Tangencial 35
Gráfica 10 Representación de una neurona Multicapa 35
Gráfica 11 Arquitectura de una red FFBP 37
Gráfica 12 Representación estructural Grafo de Huffman 48
Gráfica 13 Patrón Estructural Facade 52
Gráfica 14 Patrón Creacional Abstract Factory 53
Gráfica 15 Patrón estructural Bridge 54
Gráfica 16 Estructura Arquitectural del algoritmo 56
Gráfica 17 Diagrama de Comportamiento General 58
Gráfica 18 Diagrama de Comportamiento Paquete del proceso estadístico 61
Gráfica 19 Diagrama de Comportamiento Calculo de Niveles del Grafo 65
Gráfica 20 Diagrama de Comportamiento Proceso de Creación del Grafo 67
Gráfica 21 Asignación de Dirección hacia adelante en el Grafo 68
Gráfica 22 Asignación de Dirección hacia atrás en el Grafo 69
Gráfica 23 Diagrama de Comportamiento Proceso de Codificación 71
Gráfica 24 Cotas de Complejidad del Algoritmo 73
Gráfica 25 Arquitectura de la RNA Backpropagation para el Algoritmo 75
Gráfica 26 Descripción de las capas de la RNA 76
Gráfica 27 Descripción de la Topología de la RNA 76
Gráfica 28 Diagrama de Bloques Funcional del Prototipo 84
Gráfica 29 Prueba 1 Bits vs Trama 94
Gráfica 30 Prueba 1 Códigos Huffman vs Modificación Huffman 95

Gráfica 31 Prueba 2 Bits vs Trama 96
Gráfica 32 Prueba 2 Códigos Huffman vs Modificación Huffman 96
Gráfica 33 Prueba 3 Bits vs Trama 97
Gráfica 34 Prueba 3 Códigos Huffman vs Modificación Huffman 98
Gráfica 35 Prueba 4 Bits vs Trama 99
Gráfica 36 Prueba 4 Códigos Huffman vs Modificación Huffman 99
Gráfica 37 Prueba 5 Bits vs Trama 100
Gráfica 38 Prueba 5 Códigos Huffman vs Modificación Huffman 101
Gráfica 39 Prueba 6 Bits vs Trama 102
Gráfica 40 Prueba 6 Códigos Huffman vs Modificación Huffman 102
Gráfica 41 Prueba 7 Bits vs Trama 103
Gráfica 42 Prueba 7 Códigos Huffman vs Modificación Huffman 104
Gráfica 43 Prueba 8 Bits vs Trama 105
Gráfica 44 Prueba 8 Códigos Huffman vs Modificación Huffman 105
Gráfica 45 Iteraciones vs Tasas de Compresión 107
Gráfica 46 Comportamiento Convergente de las Iteraciones 108
Gráfica 47 Distribución de probabilidad de las tasas de compresión 112
Gráfica 48 Complejidad Computacional vs Iteraciones 113
Gráfica 49 Área útil de la linealización de la Complejidad vs Iteraciones 114
Gráfica 50 Diferencias de Complejidad de Iteración a Iteración 116
Gráfica 51 Comparación entre los códigos Huffman y el límite de Shannon 117
Gráfica 52 Área útil de trabajo respecto a las tres variables 119

INDICE DE TABLAS

Tabla 1. Calculo Complejidad Proceso Estadístico 62
Tabla 2 Calculo Complejidad Proceso Niveles del Grafo 66
Tabla 3 Calculo Complejidad Proceso Creación del Grafo 68
Tabla 4 Calculo Complejidad Proceso Propagación hacia adelante 69
Tabla 5 Calculo Complejidad Proceso Propagación hacia atrás 69
Tabla 6 Calculo Complejidad Proceso Asignación de Símbolo 70
Tabla 7 Calculo Complejidad Proceso Codificación 72
Tabla 8 Características técnicas de los videos de prueba 88
Tabla 9 Pruebas Calculo Teórico de Shannon 93
Tabla 10 Resultados de la Prueba 1 93
Tabla 11 Resultados de la Prueba 2 95
Tabla 12 Resultados de la Prueba 3 96
Tabla 13 Resultados de la Prueba 4 98
Tabla 14 Resultados de la Prueba 5 99
Tabla 15 Resultados de la Prueba 6 101
Tabla 16 Resultados de la Prueba 7 102
Tabla 17 Resultados de la Prueba 8 104
Tabla 18 Resultados de las pruebas por Iteración 106
Tabla 19 Información de segmentación de los videos de prueba 109
Tabla 20 Valores Estadísticos de referencia 110
Tabla 21 Resultados de las Pruebas en Datos Agrupados 110
Tabla 22 Función de complejidad por iteración 112
Tabla 23 Resultados de los cálculos de complejidad por iteración 113
Tabla 24 Comparación de complejidad entre las iteraciones una a una 114
Tabla 25 Comparación de complejidad entre la mejor iteración hasta la ultima 115
Tabla 26 Resultados de las pruebas en bits 116
Tabla 27 Resultados de las pruebas por complejidad 118

RESUMEN

En este trabajo se implementa un algoritmo de compresión, para ser utilizado en el proceso de codificación de la fuente de información en un sistema de telecomunicaciones. Esta implementación incluye una modificación en la estructura de datos manejada por el algoritmo de Huffman, que de forma canónica utiliza una estructura de datos abstractos tipo árbol binario para distribuir los caracteres o símbolos de la ráfaga de información a codificar, clasificándolos, dependiendo de las frecuencias relativas de aparición. Este árbol puede ser reemplazado por una red neuronal que es similar en su topología a un grafo $k - completo$ dirigido, con pesos en las conexiones o aristas, de esta manera se logra entrenar la red neuronal para que encuentre patrones en las ráfagas de información acotadas en tamaño, fragmentando el mensaje y así aumentar la tasa de compresión de la información que se quiere enviar hacia el canal, este resultado también estará enmarcado en el análisis de la complejidad temporal que requiere el algoritmo para ser ejecutado; si se logra reducir el volumen de información, significaría una mejora en el grado de servicio y desempeño de la red de telecomunicaciones aumentando la capacidad sin depender del tipo de canal ni de su ancho de banda.

PALABRAS CLAVE

Complejidad algorítmica, red neuronal, tasa de compresión, algoritmo Huffman, codificación de la fuente.

INTRODUCCIÓN

La demanda de conectividad por parte de múltiples dispositivos y el aumento del volumen de información que es transmitida por las redes de telecomunicaciones es exponencial, requiriendo canales más robustos que soporten altas tasas de transmisión, incrementando los requisitos funcionales de las redes. Este fenómeno se incrementa en las redes dispuestas para dispositivos móviles en canales inalámbricos, que es a lo que apunta la tendencia de uso; con base en lo anterior puede pensarse que la compresión de datos se hace necesaria en los escenarios de las comunicaciones; esta compresión de datos se realiza antes del envió sobre el canal, por lo general en la fuente, para tal fin se usa comúnmente el algoritmo de Huffman, con el que se codifica la información. El algoritmo de Huffman para compresión se aplica en archivos de texto, pero puede aplicarse a cualquier tipo de información si se lleva previamente a un formato digital, donde caracteres o símbolos están codificados en códigos de palabras, que mediante el cálculo de su frecuencia son distribuidos en un grafo tipo árbol binario llamado árbol de Huffman. Sin embargo, al aumentar el volumen de información a transmitir o al tratar datos muy heterogéneos, aumentará la cantidad de elementos (nodos) del grafo de Huffman, por lo tanto, aumenta la complejidad temporal y operacional del algoritmo. Este aumento genera una desventaja en los equipos que hacen uso de la conectividad, debido al incremento de recursos de hardware necesarios y que en dispositivos móviles muchas veces son limitados. También influyen las limitaciones del ancho del canal en el desempeño de la red de comunicaciones. Para tratar de encontrar una solución, las investigaciones hasta la fecha han usado comúnmente la compresión Huffman con redes neuronales utilizando pesos de red cuantificados, los cuales facilitan la transferencia de datos en ráfaga, donde la dispersión y la repetición de los pesos es común. Sin embargo, se utiliza la compresión de Huffman por separado en múltiples bloques, cada uno ejecutado por separado en hilos de procesamiento distintos, con lo que se aumenta la complejidad del algoritmo. De acuerdo a todo lo anterior se puede llegar a implementar una red neuronal que determine los patrones de compresión basado en distribuciones estadísticas y realice directamente la compresión de la información en su formato digital modificando el algoritmo de Huffman, además de esto, se puede intentar modificar la codificación Huffman convencional con una aplicación directa

de una red neuronal que se utilice como estructura de datos en las ponderaciones del grafo, para estudiar su comportamiento y desempeño.

1. PROBLEMA DE INVESTIGACIÓN

Uno de los aspectos más relevantes en la actualidad, con respecto a las telecomunicaciones es el aumento masivo de la información que se desea transmitir a través de las redes de comunicación. Otro aspecto importante es que los equipos electrónicos que hacen uso de tales redes dejaron de ser convergentes a un solo tipo de dispositivo para ser una diversa cantidad de objetos que se comunican entre sí y con sus usuarios, ambos factores desencadenan que el volumen y los tipos de información que se transmite no solo aumente en diversidad, sino también en tamaño, para dar soporte físico a esta demanda de transmisión, los canales deben hacerse más amplios y robustos. Sin embargo, esto por sí solo no es suficiente ya que debe ir trabajando en paralelo con la codificación de la información que permite comprimir los datos de su forma natural a una forma que sea más fácil de enviar a su receptor. Allí es donde se hace importante la codificación en la fuente, para lograr que el tamaño original de la información se reduzca sin tener pérdidas o distorsión. Se utilizan varios algoritmos para el proceso de codificación, el más utilizado es el algoritmo de Huffman, el cual ha tenido varias modificaciones buscando una tasa más alta de compresión, obteniendo mejoras en los resultados, no obstante, la implementación de redes neuronales directamente en el funcionamiento del algoritmo de Huffman, trazando la red como una representación del grafo de Huffman y estudiar su comportamiento, comparándolo con otros tipos de código podría llevar a encontrar una metodología para compresión de la fuente que permita la transmisión de altos volúmenes de información con una tasa más alta de compresión, mejorando el grado de servicio de redes de telecomunicaciones independientemente del canal.

¿Se puede mejorar el grado de servicio de una red de telecomunicaciones al encontrar un método de codificación de la fuente que haga compresión de la información utilizando redes neuronales directamente en la implementación del algoritmo de Huffman?

2. MARCO TEORICO Y REFERENCIAL

2.1 MARCO TEÓRICO

2.1.1 Sistemas de comunicaciones digitales

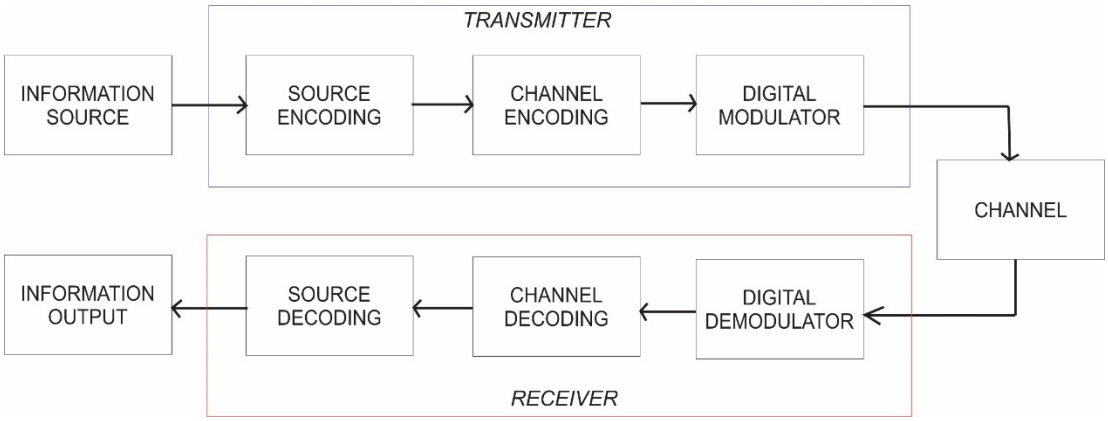
Los sistemas de comunicaciones digital son el modelo utilizado en la actualidad como forma para transmitir información de un sitio a otro, independientemente de la distancia, también independiente del medio o canal de enlace; los sistemas de comunicación digitales toman información de la fuente de origen, la transforman en su forma digital y la transportan a uno o más destinos, para luego pasarla de su forma digital a su forma original[1]; un aspecto importante de las comunicaciones digitales es el que se encarga del tratamiento del canal por el cual se transmitirá la información y los métodos con los que es tratada la información para representarla en sus diferentes formas, digitales, analógicas, este aspecto es importante ya que se debe garantizar que las pérdidas o distorsiones sean mínimas.

2.1.1.1 Elementos de las Comunicaciones Digitales

Un sistema de comunicaciones puede representarse, en su nivel más alto de abstracción como un diagrama de bloques funcionales con un orden secuencial; empezando por la fuente, que puede emitir información de cualquier tipo, como: imágenes, video, texto, etc. Esta información puede tener una forma analógica o digital dependiendo de la aplicación que la esté manejando[2]; esta información se puede representar como una señal discreta en el tiempo, conformada por un número finito de caracteres que la definen, puede estar contenida en un mensaje o un bloque de mensajes, estos son convertidos a una secuencia binaria (1,0) de variada longitud; este proceso de tomar la información y convertirla a una cadena binaria se denomina codificación de la fuente (Source Coding), en este proceso se codifica la información, también puede comprimirse o se puede hacer ambos tratamientos sobre la información original[1].

El siguiente paso es llevar la información codificada al medio de transmisión, un proceso de codificación del canal, para luego modular la señal que será enviada por el canal físico, que puede ser un medio cableado o aire en el caso que la comunicación sea inalámbrica.

Para que en el receptor la información pueda ser utilizada se sigue la misma secuencia de componentes dentro del sistema, con la diferencia que se hace de forma inversa; se empieza tomando la señal que se envió por el canal físico, pasarla a un demodulador, luego al decodificador del canal, después tendrá que decodificarse la cadena de bits de manera que esta decodificación arroje la información original y en su forma natural para que pueda ser interpretada [1].



Gráfica 1 Diagrama de Bloques Sistema de Comunicaciones Digitales

2.1.1.2 Codificación de la Fuente

Los sistemas de comunicación son diseñados con el propósito, de transmitir información desde la fuente hasta su destino, también debe considerarse muy importante el hecho de que la información puede tomar variedad de formas, un ejemplo de esto es que existen fuentes de información como la televisión (video), las estaciones de radio (voz, música) y dispositivos electrónicos como computadores, celulares etc. Estas fuentes de información pueden ser analógicas o digitales, discretas o continuas [1].

Sin importar el tipo de la fuente de información, en los sistemas de comunicación digital, las señales de información deben convertirse en su forma digital, tarea realizada por el codificador de la fuente [1], en esta parte del proceso, se debe garantizar que la información tenga la cantidad mínima de pérdidas y deformaciones, con el objetivo que, en este mismo paso en la decodificación, se obtenga una información muy parecida a la que se envió desde la fuente.

Dentro de este proceso de los sistemas de comunicaciones digitales; existen dos enfoques, con propósitos diferentes: el primero es el proceso de convertir la información sin importar su naturaleza de fuente en tramas binarias, y la segunda consiste en tomar cada trama de símbolos y transformarlos en códigos o claves[2], para representar la señal de información en un menor espacio, para luego transmitir una representación de la misma información con un menor tamaño, este último proceso es denominado compresión de la información.

2.1.1.3 Compresión de la Información.

Esta práctica en las comunicaciones digitales es muy útil en el momento de codificar el canal y modular las señales que se transmiten, ya que si se logra transmitir menos cantidad de datos, se usará menos ancho de banda del disponible en el canal y se consumirá menos energía, simplificando la capa física de la red; las tramas en codificación vienen dadas por el sistema digital con que se trabaja y no se puede alterar el número de bits arbitrariamente; por ello, se utiliza la compresión, para transmitir la misma cantidad de información en un número inferior de bits, la compresión es un caso particular de la codificación, cuya característica principal es que el código resultante tiene menor tamaño que el original[3].

La compresión de datos implica la reducción en el número de caracteres transmitidos así mismo la probabilidad de error en la transmisión se reduce, aumentando las prestaciones del sistema. Existen 2 tipos sobre los cuales se clasifican los algoritmos de compresión de datos, son compresión con pérdida y compresión sin pérdida [4].

Un algoritmo de compresión con pérdida se basa en eliminar datos de la señal para reducir el tamaño, con lo que se suele reducir la calidad [2]. En la compresión con pérdida, la tasa de bits (bit rate) puede presentar una longitud constante o variable [5]; una vez realizada la compresión, no se puede obtener la señal original, aunque sí una aproximación cuya semejanza con la original está en función del tipo de compresión. Estos algoritmos se aplican principalmente en la compresión de imágenes, videos y sonidos [6].

Los algoritmos de compresión de datos sin pérdida se caracterizan, por recuperar la señal original partiendo de la señal procesada, con lo que no se tiene una pérdida en la información; la tasa de compresión en este tipo de algoritmos es menor, que en los de compresión con pérdidas, esto debido a los métodos empleados en cada tipo de compresión [7].

2.1.1.4 Compresión sin Pérdidas

La compresión sin pérdidas es un método que permite recuperar información muy parecida a la original, este tipo de compresión es la utilizada para tratar información que no pueda tener distorsiones debido a su aplicación y uso de cara al usuario final [8].

Para realizar la compresión de datos se emplean algoritmos de codificación aplicados en la fuente, que da como resultado la información en su forma digital; existen tres tipos de algoritmos de codificación para compresión de información estos son los: adaptativos, semi adaptativos y no adaptativos.

Los algoritmos de compresión no adaptativos son aquellos que tienen asociado una tabla o diccionario de códigos, que representan las cadenas de bits que más se repiten, determinados mediante un estudio estadístico de la información, se divide en tramas de bits, a cada trama se le asigna secuencias más cortas, he irán incrementando su tamaño, son de proporcionalidad inversa al valor de la probabilidad de ocurrencia [9].

Los algoritmos semi adaptativos, empiezan revisando la cadena completa de información, para luego generar la tabla de códigos, con mucha más precisión y efectividad, logrando mayor compresión, pero empleando mayor tiempo en el procesamiento, debido a que tiene que pasar dos veces por la trama total de información [10].

Por último, los adaptativos, son algoritmos que sustituyen tramas cortas por tramas más largas que se repiten y así logra reducir el tamaño de la información.

2.1.1.5 Técnicas de Codificación y compresión de datos

Existen dos categorías principales de técnicas de compresión de datos: aquellos que utilizan métodos estadísticos y los que requieren del uso de un diccionario de prefijos [8]. Ambas técnicas son muy utilizadas, pero los esquemas basados en diccionario tienden a ser más usados para aplicaciones de compresión de ficheros en máquinas locales, mientras que las situaciones de tiempo real típicamente requieren esquemas de compresión estadísticos [11]. El motivo es que los métodos basados en diccionario tienden a ser lentos en la compresión y más veloces para descomprimir, mientras que los métodos estadísticos son igualmente veloces en ambas operaciones. En el dominio estudiado, la compresión de

los datos intenta realizarse con una aproximación de tiempo real, tomando como tiempo real la necesidad de usar el menos tiempo posible para la compresión, él envió y descompresión de los datos, para que puedan ser usados por el receptor, otra forma de explicar esto es que los métodos de compresión estadísticos se usan para comprimir datos que se envían por una red de comunicaciones [8].

Entonces en un proceso de compresión, cada bloque de datos que se comprima eventualmente será descomprimido, la velocidad combinada de compresión y descompresión puede ser tan importante como las velocidades de cada una de estas operaciones por separado [8], por lo que, los algoritmos se clasifican por su complejidad temporal, cantidad de recursos físicos que pueda consumir.

2.1.1.5.1 Métodos Estadísticos

Los esquemas estadísticos de compresión determinan el código de salida basados en la probabilidad de ocurrencia de los símbolos de entrada y son típicamente utilizados en aplicaciones de tiempo real. Como los algoritmos de compresión y descompresión tienden a ser simétricos, la compresión y descompresión usualmente requiere la misma cantidad de tiempo para completarse [12].

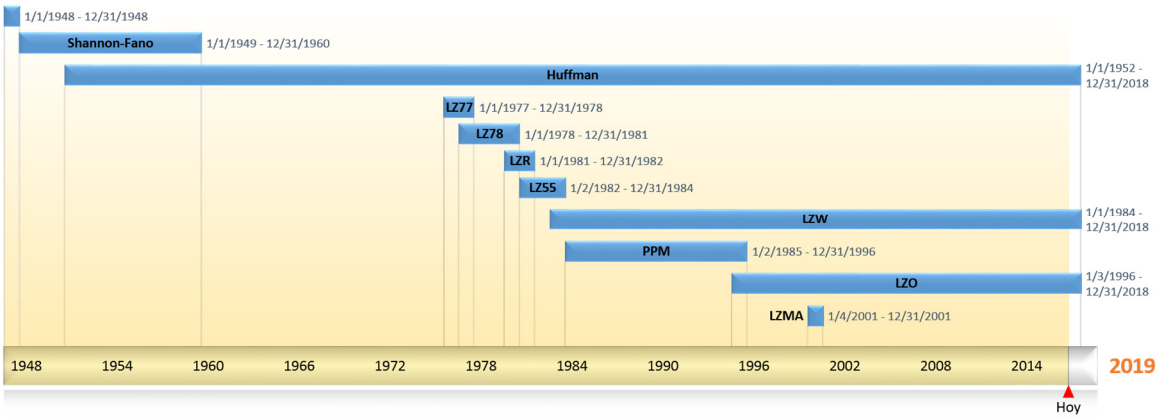
2.1.1.5.2 Métodos con uso de Diccionario de prefijos

La compresión por diccionario no usa modelos de predicción estadística para determinar la probabilidad de ocurrencia de un símbolo particular, sino que almacena cadenas de símbolos de entrada en un diccionario. Estas técnicas son típicamente usadas en aplicaciones de archivos locales, porque el proceso de decodificación tiende a ser más rápido que el de codificación, además no se tienen limitaciones de recursos como el ancho de banda [13].

2.1.2 Algoritmos de Codificación y compresión

La primera persona que trabajó sobre la información y la forma de transmitirla fue Claude Elwood Shannon; ingeniero estadounidense, padre de la moderna teoría de la información, que es una formulación matemática que analiza las unidades de información (bits) y su pérdida en los procesos de transmisión [3].

Shannon empezó a trabajar sobre el problema de la eficacia de los diferentes métodos existentes de transmisión de la información, mediante el flujo a través de hilos o cables como el aéreo, por medio de corrientes eléctricas fluctuantes o bien moduladas por la radiación electromagnética. Shannon orientó sus esfuerzos hacia la comprensión fundamental del problema y en 1948 desarrolló un método para expresar la información de forma cuantitativa [3]. Las publicaciones de Shannon en 1949 demostraron cómo se podía analizar dicha cuantificación (expresada en una magnitud que denominó bit) mediante métodos estrictamente matemáticos. Así, era posible medir la verosimilitud de la información mutilada por pérdidas de bits, distorsión de los mismos, adición de elementos extraños, etc., y hablar con precisión de términos antes vagos, como «redundancia» (elementos del mensaje que no aportan nueva información, pero se anticipan a posibles pérdidas) o «ruido» (perturbaciones en la comunicación) e, incluso, expresar el concepto físico de entropía como un proceso continuado de pérdida de información [5].



Grafica 2 Cronología de los Algoritmos de Compresión más Representativos

Esta línea de tiempo (gráfica 2)se relacionan los algoritmos más utilizados en la compresión de datos, con fines en las telecomunicaciones, desde su año de creación hasta el tiempo estimado donde se usaron con mayor aplicabilidad, se puede notar que hay varios algoritmos que hacen parte de grupos con características similares en su funcionamiento, como los LZ, que culminó su evolución en los algoritmos LZW y LZ0, también se puede ver que desde su creación en el año de 1952, el Algoritmo de compresión de Huffman, está aún vigente; una modificación y mejora del algoritmo Huffman son los llamados Huffman Adaptativos.

2.1.2.1 Codificación Huffman

La codificación de Huffman es un método de codificación y compresión de la fuente de información; es muy popular debido a que se utiliza para varios programas y plataformas, especialmente para los sistemas de comunicación [14], en donde se procesa la información, optimizando el uso de los canales disponibles. Algunos programas usan solo la codificación Huffman para representar la información en ráfagas de bits, mientras que otros lo utilizan como un paso en un proceso de compresión de varios pasos. El método Huffman es similar al método de Shannon-Fano. En general produce mejores códigos[14], y al igual que el método Shannon-Fano, produce el mejor código cuando las probabilidades de los símbolos son potencias negativas de 2. La principal diferencia entre los dos métodos es que Shannon-Fano construye sus códigos de arriba a abajo desplazándose desde la izquierda a la derecha, mientras que Huffman construye un árbol binario desde la parte inferior, construyendo los códigos de derecha a izquierda[14]. Desde su desarrollo, en 1952, por D. Huffman, Este método ha sido objeto de una investigación intensiva en la compresión de datos. El algoritmo comienza construyendo una lista de todos los símbolos del alfabeto correspondiente a un fragmento de la información que se va a codificar, el listado se organiza en forma descendente según el orden de las probabilidades de ocurrencia de cada uno de los símbolos dentro de la cadena. Luego se construye un árbol, con un símbolo en cada hoja. Esto se hace en iteraciones, donde cada una los dos símbolos con los valores más pequeños de probabilidad se seleccionan y se agregan a la parte superior del árbol parcial, se eliminan de la lista y se reemplazan por un símbolo auxiliar que representa los dos símbolos originales[14]. Cuando la lista se reduce a un solo símbolo auxiliar (que representa el alfabeto completo), el árbol es completo, siendo este último símbolo auxiliar la raíz del árbol. Luego se recorre el árbol para determinar los códigos de los símbolos.

A continuación, se muestra la estructura del algoritmo en el método de Huffman de codificación mediante su representación de una estructura de datos de tipo árbol binario.

Como primer paso se debe identificar y describir la información y su representación de la siguiente forma. Sea a_s una cadena de símbolos s, que representa en su forma natural cierta cantidad de información que se quiere codificar y comprimir mediante el método de Huffman, entonces se define el conjunto S como:

$$S: \{s_i \; es \; el \; simbolo \; que \; representa \; un \; fragmento \; de \; la \; informacion\}$$

$$S: \{s_0, s_1, s_2, s_3, \ldots\ldots\ldots s_{n-1}, s_n\}$$

Donde cada elemento del conjunto S es un símbolo que representa un fragmento o unidad minina de la información que se quiere procesar; por lo tanto, el conjunto tiene un número de elementos ($n-1$), que depende directamente del tipo, la cantidad y la naturaleza de la información[12].

La cadena a_s está conformada por un numero variable de símbolos que pertenecen al conjunto S, que se organizan indistintamente y de forma aleatoria; cada símbolo puede aparecer en a_s cero o más veces de forma que cada elemento de S tiene un valor de ocurrencia dentro de la cadena (e_{Si}); entonces se puede calcular la probabilidad de ocurrencia de cualquier símbolo s_i, a partir de la ocurrencia y la longitud de a_s [15].

$$P_{S_i} = \frac{e_{Si}}{|a_s|} \quad (2.1)$$

Por lo tanto:

$$\sum_{i=0}^{n} P_{S_i} = 1 \; (2.2)$$

Con la información descrita y organizada en un conjunto finito de símbolos que la representan y cuantificada la cantidad que representa cada símbolo dentro de la información, se procede a organizar el conjunto de probabilidades por símbolo[16]. Como cada símbolo del conjunto S tiene un valor asociado de probabilidad que varía dependiendo del número de veces que aparezca en la cadena a_s; con estos valores se genera un nuevo conjunto P_s; este conjunto se organiza con el siguiente criterio:

Sean $P_M \wedge P_N$ elementos del conjunto P_s, se debe garantizar que se cumpla que:

$$P_s: \{\ldots\ldots, P_M, P_N, \ldots\ldots\} \; si \; y \; solo \; si \; P_M \leq P_N$$

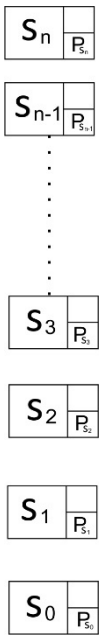
Con esta distribución se garantiza que todos los elementos de P_s se organicen dentro del conjunto de menor a mayor, teniendo como referencia el símbolo al que pertenece el valor de probabilidad[13].

El siguiente paso es construir la estructura de datos de tipo árbol binario, para lo cual se toman como referencia la relación entre los conjuntos P_s y S, previamente organizado con base a P_s.

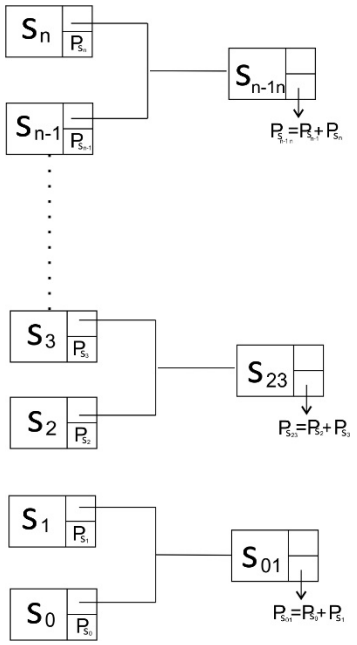
Aceptando el supuesto que la relación entre los dos conjuntos y su ordenamiento se describe con la siguiente representación:

$$P_{S_i} \rightarrow S_i \ (2.3) \ \text{[17]}$$

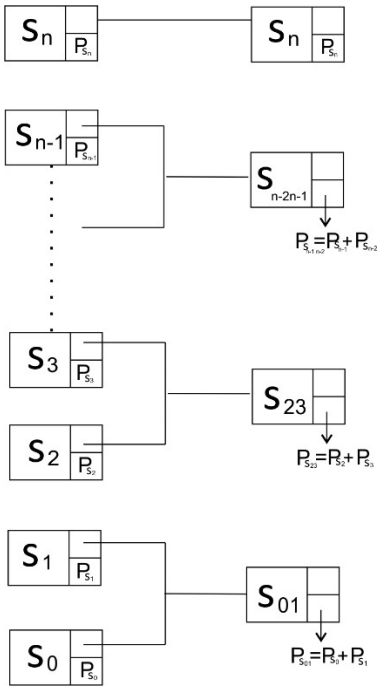
Se construirá el ultimo nivel del árbol en el cual están presentes todos los elementos del conjunto S dentro de los datos abstractos de cada nodo del árbol, la representación del nivel más profundo de la estructura se puede visualizar en la gráfica 3. Se prosigue con la creación de nodos que cumplirán la función de padres de los nodos que contienen los símbolos y agruparlos en parejas, esta distribución se hace desde el nodo con el valor más pequeño de probabilidad, la creación de estos nodo también servirá para utilizarlos como un dato auxiliar en el momento de asignar el código, este paso se verá reflejado en la gráfica 3, si el número de elementos del conjunto S es par y en la gráfica 4 si n-1 es impar[18].



Gráfica 3 Estructura del árbol de Huffman nivel con mayor profundidad

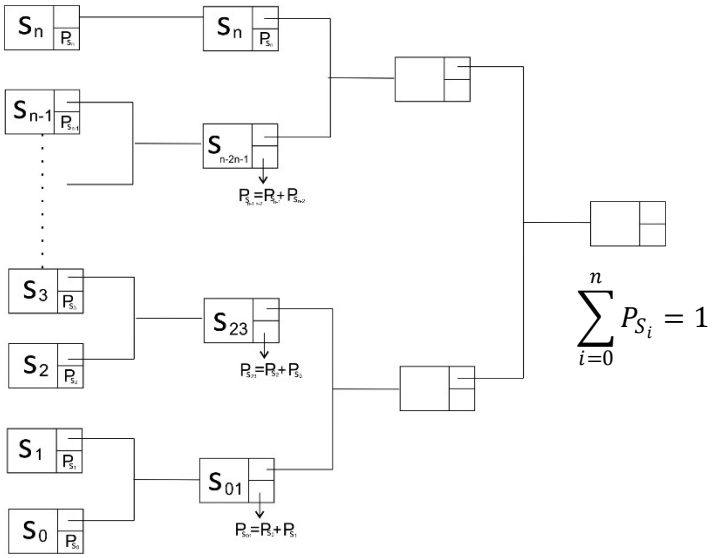


Gráfica 4 Estructura del árbol de Huffman 2° nivel con número par de elementos



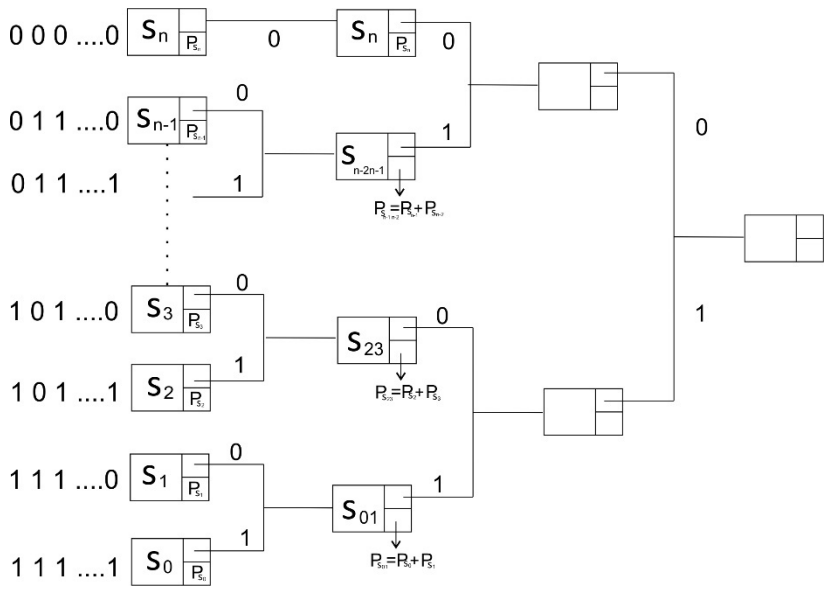
Gráfica 5 Estructura del árbol de Huffman 2° nivel con número impar de elementos

El procedimiento anterior se repite hasta completar el número de niveles de profundidad que tendrá el árbol o hasta que solo quede un nodo que se convertirá en la raíz de la estructura[16], el resultado final de las iteraciones se ve ilustrado en la siguiente figura (grafica 6)



Gráfica 6 Estructura general del árbol de Huffman

Cuando la estructura del árbol está finalizada se asignan a las ramas de este un bit (0 o 1) según el criterio de la codificación, de esta manera al realizar una búsqueda en profundidad del árbol ubicando a cada uno de los símbolos que se encuentran en el último nivel se concatena la cadena de bits que representa dado símbolo[12]; la longitud de la cadena de bits de cada símbolo no varía dependiendo la frecuencia de ocurrencia del mismo, de manera que los símbolos con mayor ocurrencia tendrán un código más corto y de manera respectiva los símbolos de menor ocurrencia un código más largo[16], la estructura del árbol con la asignación de bits en sus ramas se ilustra en la gráfica 7.



Grafica 7 *Estructura general del árbol de Huffman con código en bits*

El valor agregado de la codificación Huffman es que logra dinamizar la longitud de las cadenas de bits para cada símbolo garantizando que ningún código es prefijo de otro, de esta manera la detección y decodificación tendrá una tasa de error más pequeña. Como se ha visto con la descripción anterior del método de Huffman[19], este tiene un componente matemático bien definido a través de la teoría de conjuntos, estadística y probabilidad, las que se usan para clasificar los símbolos del fragmento de la información; es por esta razón que se puede dar las siguientes definiciones:

Tamaño Promedio del código

El tamaño promedio del código de Huffman $\bar{a}$ calculado para todos los símbolos del conjunto S se define como:

$$\bar{a} = \sum_{i=0}^{n} P_{si} l_{si} \; en \; \frac{bits}{simbolo} \quad (2.4)$$

Donde l_{si}, es la longitud en bits del código Huffman que representa el símbolo s_i y su unidad de medida es bit sobre símbolo[13].

Varianza del código Huffman

La varianza es un concepto estadístico definido para una secuencia de valores $[x_0, x_1, x_2, \ldots\ldots x_n]$, donde se mide cómo los elementos varían al calcular las diferencias entre ellos y el promedio del conjunto de la secuencia; la expresión que define matemáticamente la varianza es:

$$\sigma^2 = E(x_i - \bar{a})^2 = \frac{1}{n}\sum_{i=0}^{n}(x_i - \bar{a})^2 \quad (2.5)$$

Este concepto estadístico se aplica a la codificación Huffman ya que si se puede calcular el promedio del tamaño del código también puede calcularse la varianza, Debido a que este método puede generar códigos y arboles los cuales se pueden obtener para el mismo conjunto de símbolos[12], la varianza determina cuál de estos códigos es el más generoso de cara a la codificación en términos de recursos de procesamiento y velocidad[20].

Medida Logarítmica del código

Esta medida simplemente es una afirmación que se cumple para toda codificación Huffman, donde se puede demostrar que el tamaño del código Huffman de un símbolo cualquiera con probabilidad P_{si} es siempre menor o igual que $-\log_2 P_{si}$ [21].

$$P_{si} \leq -\log_2 P_{si} \quad (2.6)$$

Número de códigos Huffman

Dado que el código de Huffman no es único, se puede preguntar cuántos códigos diferentes se pueden obtener a partir de un conjunto dado de símbolos. Si este conjunto tiene n símbolos, el árbol de Huffman contiene $n-1$ nodos interiores o nodos distintos de la raíz, esto quiere decir que si por cada nivel del árbol se intercambian las etiquetas de sus ramas, al asignar el bit correspondiente, este produce un código Huffman diferente; de esta manera en cada rama de todos los niveles del árbol al intercambiar las etiquetas, genera un nuevo código, por lo tanto el número total de árboles de código de Huffman diferentes es igual a

$$2n-1 \quad (2.7) \text{ [16].}$$

Altura del Árbol de Huffman

La altura del árbol Huffman a veces puede ser importante porque el valor de la altura también es la longitud del código más largo que genero el árbol, por esta razón una forma de optimización del código Huffman, es buscar el árbol con la menor altura dentro de los posibles generados[12]; para encontrarlo se debe hallar la varianza que más cercana a cero se encuentre, ya que el árbol Huffman más corto se crea cuando los símbolos tienen probabilidades iguales o muy cercanas, ósea cuando la desviación estándar del conjunto de probabilidades tiende a cero[16].

2.1.2.2 Codificación Aritmética

Las implementaciones reales de la codificación aritmética son muy similares a las de codificación Huffman, aunque superan a estas últimas en la realidad, el método Huffman asigna un número entero de bits a cada símbolo, mientras que la codificación aritmética asigna un único código extenso a la cadena de entrada completa[5]. Por ejemplo, idealmente a un símbolo con probabilidad 0,4 se le debería asignar un código de 1.32 bits, pero será codificado con 2 bits usando el método Huffman[22]. Es por esta razón que la codificación aritmética tiene el potencial de comprimir datos en el límite teórico[5]. La codificación aritmética combina un modelo estadístico con un paso de codificación que consiste en algunas operaciones aritméticas. El modelo más sencillo tendría una complejidad temporal lineal de N (log (n) + a) + S n, donde N es el número total de símbolos de entrada, n es el número real de símbolos distintos, a es la aritmética a ser realizada y S es el tiempo requerido, si se necesita, para mantener las estructuras de datos internas [3].

2.1.3 Complejidad Algorítmica

Un algoritmo es un conjunto finito de acciones que dan solución a un determinado problema; en particular en las ciencias de la computación, una algoritmo es una serie de instrucciones ordenadas que logran dar solución a determinado problema de procesamiento de información; los algoritmos se pueden clasificar no solo porque lleven a cabo la tarea para la que fueron diseñados, también pueden clasificarse en cuanto a los costos de su ejecución; entendiéndose por costo, la cantidad de recursos físicos de almacenamiento y unidades de tiempo que consume para su ejecución, desde que inicia hasta que termina la

tarea, a este indicador se le conoce como medida de la complejidad del algoritmo o también como medida de la eficiencia [10].

La complejidad algorítmica permite establecer una comparación entre algoritmos equivalentes para determinar, en forma teórica, cuál tendrá mejor rendimiento en condiciones extremas y adversas[10]; siendo algoritmos equivalentes, dos algoritmos diferentes que resuelven el mismo problema; siendo la calidad de la solución otra variable a tener en cuenta, además del tiempo y el almacenamiento[23].

Para llevar a cabo una estimación de la complejidad de un algoritmo, se trata de calcular cuantas instrucciones ejecutará el algoritmo en función del tamaño de los datos de entrada[24]. Se denomina instrucción a la acción de asignar valor a una variable y a la realización de las operaciones aritméticas y lógicas, a cada una de estas instrucciones se le asigna una duración de una unidad de tiempo (1 unidad temporal)[25], ya que es difícil establecer la medida exacta del tiempo que tarda una unidad de procesamiento en ejecutar el algoritmo; también se calcula el uso de los recursos de almacenamiento temporal o permanente (memoria) que se utilizan para la ejecución del programa[10]; por lo anterior se puede decir que la complejidad algorítmica es la técnica que ayuda a definir la eficiencia de los algoritmos, respecto al uso eficiente de los recursos, éste suele medirse en función de los dos parámetros ya mencionados: el espacio, es decir, memoria que utiliza, y el tiempo, lo que tarda en ejecutarse[25]. Ambos representan los costes que supone encontrar la solución al problema planteado mediante un algoritmo. Dichos parámetros van a servir, además para comparar algoritmos entre sí, permitiendo determinar el más adecuado de entre varios que solucionan un mismo problema[12].

2.1.3.1 Complejidad Temporal

Se denomina complejidad temporal a la función $T(n)$ que mide el número de instrucciones realizadas por el algoritmo para procesar los n elementos de entrada. Cada instrucción tiene asociado un costo temporal en unidades de tiempo, normalmente medidas en microsegundos[26].

El tiempo de ejecución de un algoritmo va a depender de diversos factores como son: los datos de entrada que se le suministran, la calidad del código generado por el compilador

para crear el programa objeto[10], la naturaleza y rapidez de las instrucciones de máquina del procesador en concreto que ejecute el programa, y la complejidad intrínseca del algoritmo. Hay dos estudios posibles sobre el tiempo: Uno es el que proporciona una medida teórica (a priori), que consiste en obtener una función que acote (por arriba o por abajo) el tiempo de ejecución del algoritmo para unos valores de entrada dados. Y otro que ofrece una medida real (a posteriori), consistente en medir el tiempo de ejecución del algoritmo para unos valores de entrada dados y en una unidad de procesamiento concreta[26].

Ambas medidas son importantes puesto que, si bien la primera ofrece estimaciones del comportamiento de los algoritmos de forma independiente del dispositivo en donde serán implementados y sin necesidad de ejecutarlos[27], la segunda representa las medidas reales del comportamiento del algoritmo. Estas medidas son funciones temporales de los datos de entrada. En donde se entiende por tamaño de la entrada el número de componentes sobre los que se va a ejecutar el algoritmo[28]. Por ejemplo, la dimensión del vector de daos de entrada o el tamaño de las listas dinámicas que procesarán los datos.

La unidad de tiempo a la que deben hacer referencia estas medidas de eficiencia no puede ser expresada en segundos o en otra unidad de tiempo concreta, pues no existe un ordenador estándar al que puedan hacer referencia todas las medidas[26]. Denotaremos por $T(n)$ al tiempo de ejecución de un algoritmo para una entrada de tamaño n datos.

Teóricamente $T(n)$ debe indicar el número de instrucciones ejecutadas por un procesador idealizado. Entonces se debe buscar por tanto medidas simples y abstractas, independientes del ordenador a utilizar[27]. Para ello es necesario acotar de alguna forma la diferencia que se puede producir entre distintas implementaciones de un mismo algoritmo, ya sea del mismo código ejecutado por dos máquinas de distinta velocidad, como de dos códigos que implementen el mismo método. Esta diferencia es la que acota el siguiente principio[25].

Principio de Invarianza

Dado un algoritmo y dos implementaciones de este, $i_1, e\, i_2$, que tardan $T(n)_1\, y\, T(n)_2$ unidades de tiempo respectivamente, el Principio de Invarianza afirma que

existe una constante real $c > 0$ y un número natural n_0 tales que para todo $n \geq n_0$ se verifica que:

$$T(n)_1 \leq cT(n)_2 \ (2.8)$$

Es decir, el tiempo de ejecución de dos implementaciones distintas de un algoritmo dado no va a diferir más que en una constante multiplicativa [25].

Con esto se puede definir que un algoritmo tarda un tiempo del orden de $T(n)$si existen una constante real $c > 0$ y una implementación i_1 del algoritmo que tarda menos que $c\,T(n)$ para todo n tamaño de la entrada[25]. Dos factores para tener en cuenta son la constante multiplicativa y el n_0 para los que se verifican las condiciones, pues si bien a priori un algoritmo de orden cuadrático es mejor que uno de orden cúbico, el primero sólo será mejor que el segundo para tamaños de la entrada superiores a 200.000[25].

Es importante tener en cuenta que el comportamiento de un algoritmo puede cambiar notablemente para diferentes entradas (la aleatorización de los datos de entrada), para muchos programas el tiempo de ejecución es en realidad una función de la entrada específica, y no sólo del tamaño de ésta. Así suelen estudiarse tres casos para un mismo algoritmo: peor de los casos, mejor de los casos y caso medio[25].

El mejor de los casos corresponde a la traza (secuencia de sentencias) del algoritmo que realiza menos instrucciones. Análogamente, el peor de los casos corresponde a la traza del algoritmo que realiza más instrucciones. Respecto al caso medio, corresponde a la traza del algoritmo que realiza un número de instrucciones igual a la esperanza matemática de la variable aleatoria definida por todas las posibles trazas del algoritmo para un tamaño de la entrada dado, con las probabilidades de que éstas ocurran para esa entrada[25].

Regla General para el cálculo de Operaciones elementales (OE)

Las reglas generales para el cálculo del número de OE, siempre considerando el peor caso, se presentan a continuación; Estas reglas definen el número de OE de cada estructura básica del lenguaje, por lo que el número de OE de un algoritmo puede hacerse por inducción sobre ellas[25].

Se debe considerar que el tiempo de una OE es, por definición, de orden 1. La constante c que menciona el Principio de Invarianza dependerá de la implementación particular, pero inicialmente se supone con valor 1.

- El tiempo de ejecución de una secuencia consecutiva de instrucciones se calcula sumando los tiempos de ejecución de cada una de las instrucciones.
- El tiempo de ejecución de la sentencia CASE o selección múltiple es:

  $$T = T(c) + \max\{T(s_1), T(s_2), T(s_3), \dots\dots\dots T(s_n)\} \quad (2.9)$$

  Obsérvese que T(C) incluye el tiempo de comparación con en cada uno de los casos.
- El tiempo de ejecución de la sentencia condicional es:

  $$T = T(c) + \max\{T(s_1), T(s_2)\} \quad (2.10)$$
- El tiempo de ejecución de un bucle de sentencias es:

  $$T = T(c) + (N° \; iteraciones) * \big(T(s_1) + T(c)\big) \quad (2.11)$$

  Obsérvese que tanto $T(s_1)$ $como$ $T(c)$ pueden variar en cada iteración, y por tanto habrá que tenerlo en cuenta para su cálculo.
- Para calcular el tiempo de ejecución del resto de sentencias iterativas ciclos o saltos, serán expresadas como un bucle.
- El tiempo de ejecución de una llamada a un procedimiento o función es 1 (por llamada), más el tiempo de evaluación de los parámetros, más el tiempo que tarde en ejecutarse la función f, esto es:

  $$T = 1 + T(P_1) + T(P_2) + T(P_3) + \dots\dots\dots\dots + T(P_n) + T(f) \quad (2.12)$$

  No contabiliza la copia de los argumentos a la pila de ejecución, salvo que se trate de estructuras complejas (registros o vectores) que se pasan por valor. En este caso se contará tantas OE como valores simples contenga la estructura. El paso de parámetros por referencia, por tratarse simplemente de punteros, no se tendrá en cuenta.
- El tiempo de ejecución de las llamadas a procedimientos recursivos va a dar lugar a ecuaciones en recurrencia.

2.1.3.1.1 Cotas De Complejidad

Una vez definida la forma de calcular el tiempo de ejecución T de un algoritmo, el siguiente paso es intentar clasificar las funciones que describes el comportamiento dinámico del

algoritmo de forma que se pueda compararlas. Para ello, se van a definir clases de equivalencia, correspondientes a las funciones que se comportan de forma similar[25].

En las siguientes definiciones N denotará al conjunto de números naturales y R al conjunto de números reales.

Cota Superior, $Notación\ O$

Dada una función f, se quiere estudiar aquellas funciones g que como máximo crecen tan deprisa como f. Al conjunto de tales funciones se le llama cota superior de f y se denomina $O(f)$. Conociendo la cota superior de un algoritmo se puede asegurar que, en ningún caso, el tiempo empleado será de un orden superior al de esta cota. La definición matemática es>

Sea $f: N \to [0, \infty)$

$$O(f) = \{g: N \to [0,\infty) / \exists\ c\ \in R, c > 0, \quad \exists\ n_0\ \in N\ \cdot\ g(n) \leq c\ f(n), \quad \forall\ n\ \geq n_0\} \ (2.13)$$

Se entiende que existe un tiempo $t\ \in\ O(f)$, que indica que t está acotado superiormente por algún múltiplo de f; donde se tendrá un interesa la función f que minimiza el valor de t y que pertenezca a la función $O(f)$ [29].

Cota Inferior, $Notacion\ \Omega$

Dada una función f, se quiere encontrar las funciones g que a lo sumo crecen tan lentamente como f. Al conjunto de tales funciones se le llama cota inferior de f y se denomina $\Omega(f)$. Conociendo la cota inferior de un algoritmo se puede asegurar que, en ningún caso, el tiempo empleado será de un orden inferior al de la cota, para esto se define el conjunto de funciones como:

$$\Omega(f) = \{g: N \to [0,\infty) / \exists\ c\ \in R, c > 0, \quad \exists\ n_0\ \in N\ \cdot\ g(n) \geq c\ f(n), \quad \forall\ n\ \geq n_0\}\ (2.14)$$

En valor $t : N \to [0, \infty)$ es de orden Ω de f si t$\in\ \Omega(f)$).

Se entiende que $t\ \in\ \Omega(f)$), indica que t está acotada inferiormente por algún múltiplo de f. La mayor función de f será la que genere más interés, ya que es la función f tal que t pertenezca a Ω(f), a la que denominaremos su cota inferior[30].

Orden exacto, $Notacion\ \Theta$

Como cota asintótica exacta, se definirá como los conjuntos de funciones que crecen asintóticamente de la misma forma que el conjunto de datos de entrada. Por lo que su definición seria:

Sea $f: N \rightarrow [0, \infty)$

Se define el conjunto de funciones de orden Θ de f como:

$$\Theta(f) = O(f) \cap \Omega(f) \quad (2.15\ a)$$

$$\Theta(f) = \{g: N \rightarrow [0,\infty) / \exists\ c, d \in R, c, d > 0, \quad \exists\ n_0 \in N \cdot c\ f(n) \leq g(n) \leq d\ f(n), \quad \forall\ n \geq n_0\} \quad (2.15\ b)$$

Intuitivamente, $t \in \Theta(f)$ indica que t está acotada tanto superior como inferiormente por múltiplos de f, es decir, que $t\ y\ f$ crecen de la misma forma[25].

2.1.3.2 Complejidad Espacial

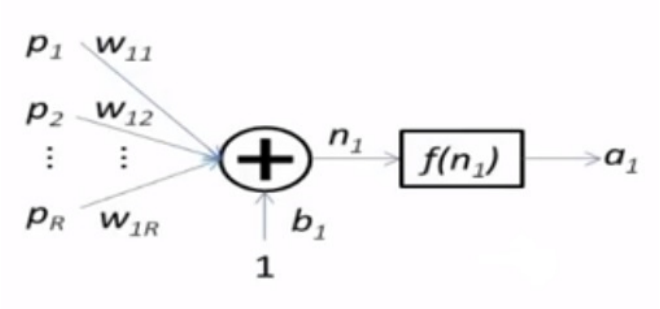
La complejidad espacial de los algoritmos hace referencia a la cantidad de memoria de procesamiento requerida en tiempo de ejecución, la complejidad espacial, en general, tiene mucho menos interés. El tiempo es un recurso mucho más valioso que el espacio. Sin embargo, el cálculo de la cantidad de memoria requiere un algoritmo esta implícitamente dentro del cálculo de la complejidad temporal, ya que, dentro de las operaciones elementales la asignación de variables es la única que consume una unidad de almacenamiento[10].

2.1.4 Redes neuronales artificiales

Una red neuronal artificial (RNA) puede definirse como un sistema de procesamiento de información compuesto por un determinado número de elementos de procesamiento (neuronas), conectados entre sí a través de canales de comunicación denominados sinapsis [31]. Estas conexiones establecen una estructura jerárquica y permiten la interacción, representando las abstracciones del mundo real, estos sistemas computacionales tratan de emular al sistema nervioso biológico.

2.1.4.1 Elementos de las RNA

El primer componente de una neurona artificial son las entradas P_R , y son representados por la letra P ya que también se conocen como patrones, las cuales se multiplican por los pesos sinápticos W_{ij}, una entrada especial b_1 que representa la polarización de la neurona; la suma de estas entradas multiplicadas por los pesos sinápticos más la polarización da como resultado la entrada neta de una neurona, esta pasa por la función de activación que es la encargada de decidir cuando la neurona se activa o no, dando como salida a_i [32]; a continuación se muestra en la gráfica 7 la representación básica de una neurona artificial.



Gráfica 8 Representación básica de una neurona artificial

[33]

Donde:

$$P : Vector\ de\ entradas$$

$$W_{ij}: Vector\ de\ pesos\ sinapticos$$

$$b : Valor\ de\ la\ polarización$$

$$n_i: Entrada\ neta$$

$$f(n_i): Función\ de\ activación$$

$$a_{ij}: Salida\ (axón)$$

Se puede representar los valores de las entradas y los pesos sinápticos como un vector y de esta manera evaluar mediante producto punto de matrices el valor de la salida neta[34].

Entonces:

$$w_{ij} = \begin{bmatrix} W_{11} \\ W_{12} \\ \vdots \\ W_{1R} \end{bmatrix} \; p = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_R \end{bmatrix} \quad (2.16\,a)$$

$$n_i = w_{ij}^T P + b \quad (2.16\,b)$$

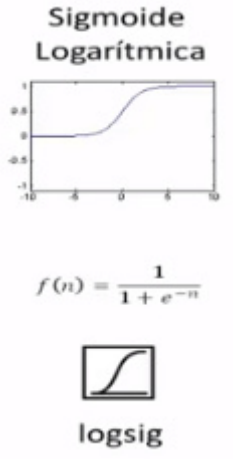
La salida de la neurona está representada matemáticamente por:

$$a_{ij} = f\left(w_{ij}^T P + b\right) \quad (2.16\;\; c)$$

2.1.4.2 Funciones de Activación

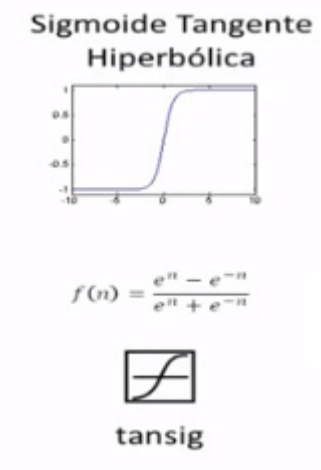
Las funciones de activación en las RNA son las encargadas de mapear, escalar o limitar el dominio de las entradas por los pesos sinápticos hacia un rango especifico a la salida de la función, las funciones de activación que se utilizan más comúnmente son[35]:

Función Sigmoide



Esta función tiene una pendiente continua, lo que facilita el entrenamiento de RNA multicapa[31].

Grafica 8 Función Sigmoide Logarítmica

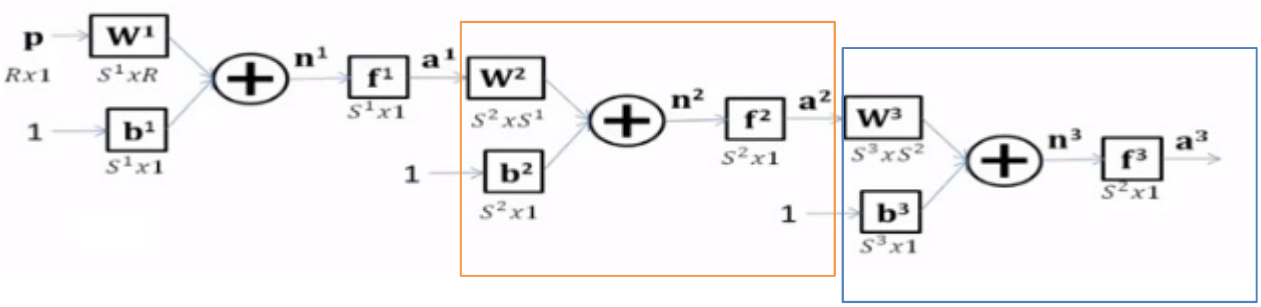


Es una variación de la anterior, esta tiene valores de activación en 1 y de desactivación en -1[31].

Grafica 9 Función Sigmoide Tangencial

2.1.4.3 RNA Multicapa

Una red neuronal multicapa es la unión de varias redes monocapa en serie, lo que quiere decir es que la salida de la capa uno está conectada a la entrada de la capa dos y así respectivamente si la RNA tiene más de una capa[36], la siguiente Gráfica (10) muestra la representación de una RNA multicapa:



Gráfica 10 Representación de una neurona Multicapa

La representación matemática sería igual que la de una red monocapa con la diferencia que las funciones de capa a capa se convertirán en composición de las funciones anteriores[35].

2.1.4.4 RNA Feed Forward Back Propagation (FFBP)

El método de entrenamiento para una red neuronal multicapa con propagación hacia atrás del error medio cuadrático[37], es la asociación que existe entre los patrones de entrada a la misma y las clases correspondientes que acotan la salida neta de la red, utilizando más niveles de neuronas que los que se usan en una red multicapa o una red perceptrón[38].

Este método es conocido como Backpropagation, que es un tipo de red con aprendizaje supervisado, el cual emplea un ciclo propagación-adaptación de dos fases[39].

Una vez aplicado un patrón de entrenamiento a la entrada de la red, este se propaga desde la primera capa a través de las capas subsecuentes de la red, esta fase del entrenamiento es la propagación hacia adelante (Feed Forward), hasta generar una salida, la cual es comparada con la salida deseada, con la que se calcula una señal de error para cada una de las salidas en las capas ocultas de la red, a su vez esta es propagada hacia atrás, empezando de la capa de salida, hacia todas las capas de la red hasta llegar a la capa de entrada, con la finalidad de actualizar los pesos sinápticos de cada neurona, para hacer que la red converja a un estado que le permita clasificar correctamente todos los patrones de entrenamiento[37].

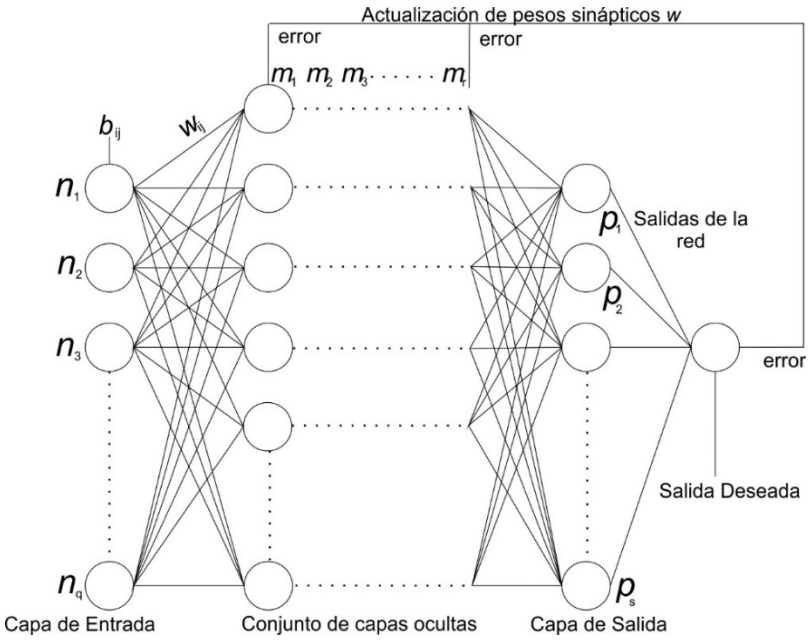
La importancia de la red Backpropagation, consiste en su capacidad de auto adaptar los pesos sinápticos de cada una de las neuronas que aportan a la salida neta de la red, utilizando el aporte relativo al error total[40]. De esta manera logra aprender la relación entre los patrones de entrada y las salidas deseadas, la capacidad de generalización de las salidas satisfactorias a patrones que no se han visto en el entrenamiento, elevando la capacidad de clasificación y reconocimiento de patrones diferentes asociados al contexto del problema abordado en la fase de entrenamiento[41].

2.1.4.4.1 Estructura de una red FFBP

Este tipo de red se basa en el modelo básico de una red multicapa y al igual que estas la red Backpropagation se compone de una capa de n neuronas, una capa de salida con m neuronas y por lo menos una capa oculta[39]. Cada una de las neuronas de una capa exceptuando la capa de entrada, recibe señales de todas las neuronas de la capa anterior

y envía las señales de salida a la capa posterior excepto la capa de salida a la red y no existen sinapsis hacia atrás (Feed back) ni laterales entre las neuronas de una misma capa.

En la gráfica (11), se representa una red FFBP con q neuronas en la capa de entrada, m capas ocultas y s neuronas en la capa de salida[39].



Gráfica 11 Arquitectura de una red FFBP

2.1.4.4.2 Aprendizaje Backpropagation

La aplicación del algoritmo de aprendizaje de este tipo de redes consta de dos fases, como ya se expuso anteriormente; la primera fase es la de propagación hacia adelante, durante esta fase se envía a la red el patrón de entrada y propagado a través de las capas ocultas hasta llegar a la capa de salida; al obtener los valores de las salidas de las neuronas de la capa de salida, se inicia la segunda; esta empieza con una comparación entre los valores de salida y la respuesta esperada, de esta manera se calcula el error; se propaga el valor del error hacia la capa anterior (retro propagación), se ajustan los pesos sinápticos de la capa con respecto al error; el error se calcula de nuevo y se retro propaga hacia la capa anterior de la que ya fue actualizada[39]; este proceso continua iterativamente hasta llegar

a la primera capa, de esta manera se acotan los pesos sinápticos de la red con cada patrón de entrada[37].

El algoritmo Backpropagation requiere neuronas con funciones de activación de tipo continuo y por lo tanto diferenciable en cualquier punto para de esta manera usar en método de descenso del gradiente, por lo general esta función será sigmoidal logarítmica o tangencial[37].

2.1.4.4.3 Algoritmo Backpropagation

- Inicializar los pesos de la red (w) con valores aleatorios pequeños.
- Condición de Break para el ciclo de propagación o número de iteración dado.
- Se presenta un patrón de entrada, $(t_1, t_2, t_3, \dots\dots, t_q)$ y se específica la salida deseada que debe generar la red $(p_1, p_2, p_3, \dots\dots, p_q)$.
- Se calcula la salida actual de la red, para ello se presentan las entradas a la red y se va calculando la salida que presenta cada capa hasta llegar a la capa de salida.
    - Se determinan las entradas netas para las neuronas ocultas procedentes de las neuronas de entrada.
    - Se aplica la función de activación a cada una de las entradas de la neurona oculta para obtener su respectiva salida.
    - Se realizan los mismos cálculos para obtener las respectivas salidas de las neuronas de la capa de salida.
    - Determinación de los términos de error para todas las neuronas:
        - Cálculo del error (salida deseada–salida obtenida).
        - Obtención de la delta (producto del error con la derivada de la función de activación con respecto a los pesos de la red).
- Actualización de los pesos. Se emplea el algoritmo recursivo del gradiente descendente, comenzando por las neuronas de salida y trabajando hacia atrás hasta llegar a la capa de entrada.
- Se cumple la condición de break (error mínimo o número de iteraciones alcanzado)

2.2 MARCO REFERENCIAL

2.2.1 Sistemas de comunicaciones digitales, Codificación de la Fuente

El campo de las comunicaciones digitales es popular en la actualidad, debido a las prestaciones que ofrece y su uso cotidiano, el cual es transparente al usuario pero que contiene la estructura medular de las comunicaciones a nivel mundial; en esto radica el interés de las investigaciones en cada uno de los bloques funcionales de estos sistemas. Este trabajo se referencia en particular en el proceso de codificación de la fuente que es el encargado de tomar la información y representarla en cadenas de bits, para que pueda ser transmitida; buscando dos características especiales; la primera es que, sin importar el tamaño de almacenamiento, la información pueda representarse en cadenas lo más cortas posibles (compresión) y la segunda, que el código generado pueda decodificarse de tal forma que genere la menor cantidad de errores posibles, para no comprometer la integridad de la información (codificación sin pérdidas). Estas dos características son inherentes al método de codificación y definirán el desempeño de esta parte del sistema. A continuación, se resumen algunas de las investigaciones en las que se enmarca este trabajo.

Dentro de los procesos de los sistemas de comunicación digital se encuentran los de codificación y modulación, en este aspecto existen investigaciones recientes como la titulada “ *“Codificación y modulación de Fuente y canal conjunta mediante códigos de entrelazado con fuentes de Márkov”*, que plantea un esquema de codificación de canal de fuente conjunta distribuida basado en un código de unidad asistida por URC, donde se considera la entrada de dos fuentes de Márkov distintas, donde se supone que una de las fuentes está perfectamente descodificada y está disponible en el receptor como información lateral a usar como un diccionario de prefijos. Como resultado de la implementación y posterior simulación, se halló que este enfoque mejora el proceso de codificación de la fuente y del canal en lo que al tiempo se refiere y al espacio de almacenamiento, ya que la hipótesis plantea que se puede usar este método para unir los bloques de fuente y de canal del sistema en uno solo pasando directamente a la modulación[42].

Otros artículos proporcionan estudios de varias técnicas de codificación sin pérdidas y comparan su rendimiento y eficiencia utilizando la complejidad de tiempo y espacio. La compresión de datos es en sí misma un gran campo de la informática y es utilizado en la

reducción del tamaño de los datos mediante la eliminación de caracteres redundantes o codificación de los bits en datos.

En estos trabajos se muestran técnicas de compresión de datos sin pérdida, para la compresión de varios archivos de formatos diferentes como texto, imagen y audio. Se comparan los métodos de Huffman, como una codificación que genera cadenas de longitud variable para cada símbolo. También LZW, que es una técnica basada en diccionario y codificación Shannon-Fano, que se utiliza para generar código binario basado en la frecuencia de cada símbolo. De esta manera se calcula la relación de compresión media, Factor de compresión y tiempo de compresión para varios archivos multimedia, luego se hace el análisis de los resultados de cada una de las técnicas de compresión de datos anteriormente mencionadas[43].

2.2.2 Complejidad Algorítmica

El tema de la predicción y evaluación de algoritmos es de suma importancia en este trabajo investigativo, ya que esta técnica es la base conceptual y teórica para enfrentar la decisión de qué algoritmo es realmente eficiente para mejorar el proceso de codificación, visto desde la compresión de datos. Siendo esta última la meta final del proceso. Las características a evaluar son la eficiencia, el tiempo y el almacenamiento utilizado para el procesamiento, estas son fundamentales en el momento de dar una propuesta computacional como solución al problema de optimizar la codificación de la fuente y así mejorar la transmisión de datos.

Por lo anterior es importante indagar y enmarcar este trabajo, en temas actuales e investigaciones que abran puertas en el tema de valoración de la complejidad algorítmica.

Existen trabajos que proponen el uso de una aplicación con el fin de evaluar las vulnerabilidades que puede tener un algoritmo en el exceso de utilización de recursos de tiempo y espacio en ejecución; la investigación que da como producto la aplicación mencionada, concluye que no es posible construir un modelo completamente automatizado para calcular la complejidad algorítmica a un proceso y por consiguiente encontrar las vulnerabilidades que pueda tener el algoritmo y de esta manera optimizarlo; el problema radica en que las entradas al algoritmo son aleatorias dentro del rango de variables con las que trabaja el algoritmo y tanto el rango como el tipo de variable puede aleatorizarse, otra

dificultad es la capacidad de determinar la arbitrariedad y/o tiempo de uso de una estructura de control, especialmente los ciclos. La propuesta entonces se basa en un análisis algorítmico dinámico sobre varias herramientas que se integran para lograr una semiautomatización del cálculo de complejidad[44].

2.2.3 Codificación con el método de Huffman

La compresión de datos es un uso generalizado en las telecomunicaciones. La codificación de Huffman es la técnica más popular para este fin, esta técnica genera códigos sin prefijo dentro de tramas de bits que representan la información. Es un algoritmo eficiente en el campo de la codificación de fuente. Produce el número más bajo posible de símbolos de código de un único símbolo de fuente. La codificación de Huffman es la técnica de compresión sin pérdida más ampliamente utilizada. Es por esta razón que este trabajo se enmarca en las investigaciones recientes que tratan sobre la codificación de la fuente de sistemas de comunicaciones digitales utilizando el código Huffman; la revisión bibliográfica de este tema, tiene caminos amplios ya que como es un código con mucha utilización las variaciones y asociaciones con otras técnicas son varias y estas están reflejadas en investigaciones y propuestas que se realizan, buscando mejoras en la codificación; a continuación se muestran artículos de investigación, que son los más representativos en el contexto de este documento.

En uno de estos artículos se detalla un nuevo enfoque para la compresión de enteros sin pérdida que se puede utilizar para extender y mejorar varios Métodos de compresión de datos sin pérdida dinámicos existentes. La nueva propuesta, denominada como Delta-Huffman, utiliza el código Delta de Elias como una representación decodificable única del alfabeto infinito de enteros ilimitados y utiliza esta representación para permitir la aplicación de la codificación dinámica de Huffman en el conjunto de enteros codificados de Delta. El método puede extenderse a combinaciones de otras técnicas de codificación de enteros con algoritmos de codificación de símbolos dinámicos adicionales[45].

Otra investigación plantea como la compresión de datos se utiliza como técnica en un sistema de procesamiento de señales digitales, y uno de los enfoques más eficientes es la codificación de Huffman. Este artículo propuso un nuevo algoritmo parcialmente paralelo al método Huffman, que combina la construcción de árboles de Huffman con la codificación

de desplazamiento, para acelerar la generación de la tabla de codificación de Huffman en el codificador. Los resultados del análisis teórico y la simulación posterior al diseño muestran que, debido a la característica parcialmente paralela al método Huffman, el algoritmo propuesto puede mejorar aproximadamente el 75% de la velocidad de codificación de Huffman, mientras que es adecuado para implementarlo en una matriz de compuertas reprogramables “FPGA”[46].

Por último, un trabajo de investigación indaga y profundiza en algunas limitaciones que surgen en la codificación de Huffman. Este método produce un código de pocos bits para un símbolo que tiene una alta probabilidad de ocurrencia y un gran número de bits para un símbolo que tiene una baja probabilidad de ocurrencia. En lugar de esto, en la codificación de Doublé Huffman, cuando se haya generado la palabra clave del símbolo, se comprimirá sobre una base binaria. A través de esta técnica se logrará un mejor resultado. En este artículo se discute la técnica de codificación de Huffman y la codificación de Doble Huffman y compara su análisis de rendimiento[16].

2.2.4 Redes Neuronales Artificiales

Las redes neuronales artificiales son utilizadas en muchos campos de la ingeniería, en especial en las aplicaciones enfocadas a desarrollos tecnológicos; es así como esta y varias herramientas de inteligencia computacional se pueden usar y adaptar a la solución de problemas en diversos contextos; en este trabajo se toma como referencia la capacidad que tienen las redes neuronales para dos funciones específicas. La primera es que las RNA mediante aprendizaje pueden aprender a clasificar, organizar y reconocer patrones, para este caso, patrones en las cadenas de información que se van a codificar; la segunda funcionalidad en la representación es la capacidad de procesamiento mediante una representación de grafo, con nodos como unidades individuales(neuronas).

En el documento “*Convolutional Neural Network Based Synthesized View Quality Enhancement for 3D Video Coding”, “Mejora de la calidad en la codificación de video 3D utilizando redes neuronales convolucionales”,* se propone un enfoque para mejorar la eficiencia de la codificación de imágenes de video para ser comprimidas y transmitidas sin pérdidas; para esto se propone un método de mejora de la calidad de la vista sintetizada basada en redes neuronales convolucionales (CNN), para la codificación de video de alta

eficiencia 3D (HEVC). En primer lugar, se trabaja sobre la eliminación de la distorsión en la vista sintetizada, se formula como una tarea de restauración de la imagen con el objetivo de reconstruir la imagen sintetizada sin distorsión latente mediante el remplazo de los patrones identificados por las CNN. En segundo lugar, los modelos CNN aprendidos se incorporan al códec HEVC 3D para mejorar el rendimiento de la síntesis de la vista tanto para la optimización de la síntesis de la vista (VSO) como para la vista sintetizada final, donde las distorsiones geométricas y de compresión se consideran de acuerdo con las características específicas de la vista sintetizada. En tercer lugar, se propone un nuevo multiplicador de Lagrange en la función de costo de distorsión de la velocidad (RD), el cual se deriva para adaptar el proceso VSO basado en CNN y así abarcar un mejor rendimiento de codificación de video 3D. Los amplios resultados experimentales muestran que el esquema propuesto puede eliminar de manera eficiente los errores en la imagen sintetizada y logra reducir la tasa de bits del 25,9% y el 11,7% en términos de índice de relación señal/ruido (PSNR) y de similitud estructural (SSIM), que supera significativamente los métodos que se usan actualmente[47].

El artículo resumido aquí, es de vital importancia y enmarca uno de los objetivos de este trabajo que es lograr realizar una codificación de la información mejorando la taza de compresión sin pérdidas, reduciendo el número de bits de error para mejorar la transmisión y para ello se utiliza un modelo de RNA, que logra clasificar y reconocer los patrones faltantes para reemplazarlos en las imágenes procesadas; esto demuestra que la idea de utilizar modelos de RNA, para los algoritmos de codificación, se está investigando en profundidad y arrojando buenos resultados en las pruebas.

2.2.5 Algoritmo de Huffman y las aplicaciones con redes neuronales u otros métodos de Inteligencia computacional

La utilización del algoritmo de Huffman para realizar la codificación de información es común en las aplicaciones de transmisión de datos, por esta razón, son varias las investigaciones que han abordado los enfoques de compresión de datos en la codificación, no solo con este algoritmo sino con varios de los que ya se hizo mención anteriormente, la

tasa de compresión de estos algoritmos es cuantificada por el tamaño de la información a la entrada y a la salida del sistema, esto indica el desempeño del proceso de codificación.

Al tratarse de un algoritmo que utiliza procesos estocásticos y probabilísticos, este ejecuta procesos iterativos de reconocimiento de cadenas similares en la ráfaga de información, por esta razón se han hecho estudios de cómo utilizar herramientas de inteligencia artificial para trabajar con el algoritmo de Huffman. Sin embargo, las investigaciones alrededor de este tema tratan la compresión de datos como bloques independientes donde en hilos separados se ejecuta el algoritmo de codificación y en otro la compresión de la cadena. A continuación, se mencionan algunos de los trabajos más recientes y cercanos al tema de investigación de este trabajo:

En varias investigaciones se hace referencia a una implementación que optimiza la codificación de video 3D, para mejorar la eficiencia en la codificación de la fuente de información, en este caso imágenes; se utiliza una red neuronal convolucional (CNN), basado en el método de mejora de la calidad de la vista sintetizada, para codificación de video de alta eficiencia 3D (HEVC), se propone, en primer lugar, la eliminación de la distorsión en la vista sintetizada, se formula como una tarea de restauración de la imagen, con el objetivo de reconstruir el latente de la imagen sintetizada sin distorsiones, en donde se utiliza la RNA como predictor y corrector de errores, como pérdida de la información[47].

También existen trabajos que implementan una RNA en particular de aprendizaje profundo (DNN) que producen alta precisión en los resultados de aplicaciones algorítmicas, sin embargo, incurren en altos costos de complejidad computacional y requisitos de memoria, la propuesta que ofrece es una estructura de la RNA trabajando en forma paralela con el algoritmo de Huffman haciendo mejoras para la ejecución en un móvil con recursos limitados llegando a una compresión de datos mejorada, no obstante se trabaja el algoritmo de Huffman independiente de la RNA [48].

Otras investigaciones realizan una evaluación y comparación de desempeño de los métodos de compresión de Huffman y Shannon- Fano, trabajando sobre dispositivos móviles y su interacción en la transmisión de información de cara a la red a la que se conecte el dispositivo, siendo estas redes de wifi o red móvil 3G, también se realiza el

análisis del uso de estas metodologías de compresión en el momento de utilizarlas por aplicaciones que funcionan en estos dispositivos [14].

3. DESARROLLO DE LA INVESTIGACION

3.1 PROPUESTA DEL MODELAMIENTO MATEMATICO DEL ALGORITMO DE HUFFMAN MODIFICADO

El algoritmo de Huffman se basa en esencia en la construcción de un árbol binario, en el cual se guardan las rutas donde cada símbolo del alfabeto es almacenado; la utilización de esta estructura para llevar a cabo esta función dentro del algoritmo es la primera parte que puede describirse de una manera diferente mediante conceptos de matemáticas discretas, en particular con la teoría de grafos.

3.1.1 Modelado del Grafo de Huffman

Los grafos se utilizan en muchas ocasiones para modelar problemas que pertenecen al campo de las telecomunicaciones, por ejemplo, para describir la arquitectura de una red, en la cual cada nodo hace referencia a un componente del sistema; entonces el modelamiento vendría siendo la construcción del modelo como un proceso consistente en el cual se describe cuáles son las características o aspectos que representan el problema y la solución a este.

Por lo anterior lo que se quiere conseguir es representar el árbol de Huffman generado por el algoritmo, por medio de un grafo, que será una estructura TDA (Tipo Abstracto de Dato), para ello se utilizará la siguiente definición de grafo:

Un grafo $G = (N, A, f)$, *es una estructura que consta de un conjunto no vacío de nodos, puntos o vértices identificado con la letra* $N{:}\{\}$; *un conjunto* $A{:}\{\}$ *de aristas y una correspondencia* f *del conjunto de aristas* A *en un conjunto de pares ordenados o no de* N.

Con la definición anterior se puede llegar a concluir que el árbol de Huffman también es un grafo conexo no dirigido. Es no dirigido ya que, para realizar la búsqueda dentro de él, no requiere que se lleve el flujo del recorrido en una dirección específica ya que cada nodo solo tiene dos posibles rutas; este tipo de búsqueda se denomina búsqueda en amplitud. Este tipo de búsqueda en ciencias de la computación es un algoritmo de búsqueda no informada utilizado para recorrer o buscar elementos en un grafo, frecuentemente sobre árboles. Intuitivamente, se comienza en la raíz y se exploran todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

La forma en la que funciona esta búsqueda ahora desde la raíz consiste en visitar todos los nodos por niveles partiendo de la izquierda hacia la derecha, una vez que se termina con el segundo nivel se pasa al tercero y así sucesivamente hasta recorrer todo el árbol o grafo.

De lo anterior se puede extraer un concepto importante, este término es el nivel; en los árboles los niveles, son la cantidad de generaciones (padres-hijos) que tiene el árbol; este concepto se puede extrapolar a un grafo distinto de un árbol, contando el número de aristas en cada par de nodos, ya que puede existir más de un camino de un nodo a otro.

3.1.1.1 Elementos matemáticos del Modelo

Teniendo en cuenta las anteriores consideraciones, se procederá a proponer la siguiente modelización matemática de un grafo conexo dirigido que represente un árbol de Huffman con cualquier alfabeto de entrada:

- Conjunto de Símbolos de entrada o alfabeto.

$$S: \{s_i \; es \; el \; simbolo \; que \; representa \; un \; fragmento \; de \; la \; informacion\}$$

$$S: \{s_0, s_1, s_2, s_3, \dots \dots \dots \dots s_{n-1}, s_n\} \quad (3.1)$$

Se tiene a la entrada del algoritmo un conjunto de símbolos que conforman el alfabeto que representa la información a la entrada del codificador, este conjunto de símbolos varía su tipo dependiendo de la naturaleza de la información.

De la definición del conjunto S se extrae:

- El número de símbolos individuales del conjunto.

$$n : es\ el\ número\ de\ símbolos\ o\ elementos\ del\ conjunto\ S$$

El comportamiento del grafo varia si el valor de n es un número par o impar; esto debido a que, si el número de símbolos es par, el grafo generado será de un grafo simple completo; por el contrario, si el número de símbolos es impar, el grafo será un grafo incompleto y por tanto se tendrá que completar mediante un nodo auxiliar, para describir este procedimiento se dará la definición de grafo completo.

Un grafo G es completo si cada par de vértices está conectado por una arista. En donde el grafo completo compuesto por el conjunto N vértices tiene $\frac{N(N-1)}{2}$ aristas, y se denota como K_N.

Tomando como referencia la definición anterior y la cantidad de símbolos del conjunto S, representado por n, puede calcularse el número de niveles que tendrá el grafo o lo que es su análogo la altura del árbol.

- El número de niveles del grafo N_N.

$$N_N = \begin{Bmatrix} \log_2 n + 1 & si\ n\ es\ par \\ \log_2(n+x) + 1 & si\ n\ es\ impar \end{Bmatrix} \quad (3.2)$$

Donde x es el numero de nodos auxiliares que se deben agregar para convertir el grafo en un grafo completo.

Ahora se debe calcular el número de nodos que se necesitarán para almacenar el conjunto de símbolos del conjunto S, sumando los nodos auxiliares que se necesitarán para completar el grafo.

- Número de Nodos para el conjunto S: N_S

$$N_S = 2^{N_N} \quad (3.3)$$

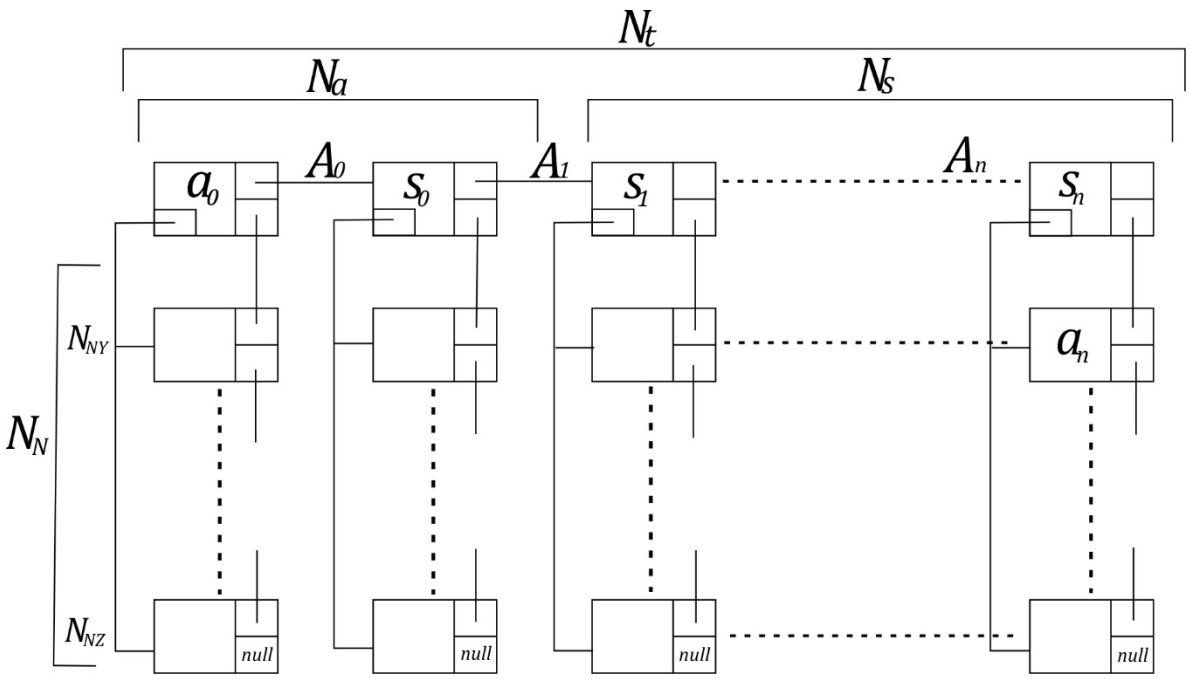
$$por\ tanto\ el\ numero\ de\ nodos\ auxiliares\ N_a = 2^{N_N} - n$$

Por último, se puede hallar el número total de nodos que conformarán el grafo así:

- Número total de nodos del grafo, N_T

$$N_T = \sum_{i=0}^{N_N} 2^i + 1 \quad (3.4)$$

En la siguiente figura (gráfica 12) se muestra la representación gráfica de la estructura general de un grafo de Huffman, esta estructura será construida por el algoritmo de Huffman modificado.



Gráfica 12 Representación estructural Grafo de Huffman

La anterior figura representa un grafo completo G, con N_a Nodos auxiliares, N_S nodos que almacenan el conjunto de n símbolos del alfabeto, A_n aristas de conexión entre nodos y N_N niveles de profundidad.

3.1.1.2 Conjunto A del grafo de Huffman

La cantidad de memoria de almacenamiento y memoria de procesamiento que usa cualquier tipo de algoritmo, se basa en la cantidad de datos que necesita y el número de referencias que requiere para procesar esos datos, el algoritmo propuesto no es la excepción a la utilización de estos recursos que influyen directamente en la complejidad algorítmica, pero el modelamiento presentado busca disminuir la cantidad de memoria de

almacenamiento remplazándola por referencias en ejecución, lo que quiere decir es que se construirá el grafo de Huffman de tal manera que la cantidad de nodos necesarios N_T debe ser mucho menor que la cantidad de referencias a cada nodo, lo que vendría siendo el conjunto de caminos de un nodo a otro, o sea el conjunto A o conjunto de aristas del grafo, por lo tanto:

$$N_T \ll \boldsymbol{A} \quad (3.5)$$

Para lograr esto se deben establecer las reglas de generación tanto del número de nodos N_T y el número de aristas A con respecto a la relación entre estos dos conjuntos f, de tal manera que la cantidad de elementos de A sea independiente de la cantidad de símbolos, pero este en función de la cantidad de nodos de un nivel específico del grafo.

Entonces se debe considerar la cantidad de referencias que cada nodo va a tener hacia otros nodos del árbol; este valor al igual que la cantidad de niveles del grafo depende del número de símbolos que tiene el conjunto S, pero no de su cantidad, sino de su naturaleza, es por esto que, para definir el destino, la cantidad de referencias y la ubicación de cada nodo, se depende del valor de n.

- Niveles intermedios de profundidad del grafo (Y)

  $Y: es\ cualquier\ valor\ entre\ 0\ y\ N_N\ e\ identifica\ cualquier\ nivel\ del\ grafo\ que\ no\ es\ el\ primero\ ni\ el\ último$

- Último nivel de profundidad del grafo (Z)

  $Z: Representa\ un\ número\ que\ ubica\ el\ último\ nivel\ de\ profundidad\ del\ grafo$

- Nivel Inicial del grafo (X)

  $X: toma\ valor\ de\ ubicacion\ 0, representa\ que\ es\ el\ primer\ nivel\ del\ grafo, además\ en\ este\ nivel\ se\ almacenan\ los\ símbolos$

Estas variables se relacionan entre sí con respecto a X , de la siguiente manera:

$$Y = 2X \quad (3.6\ a)$$

$$Z = 2X + 1 \quad (3.6\ b)$$

$$X = \begin{Bmatrix} \frac{Y}{2} \; si \; las \; referencias \; en \; A \; son \; pares \\ \frac{Z-1}{2} \; si \; las \; referencias \; en \; A \; son \; impares \end{Bmatrix} \quad (3.6\; c)$$

3.2 DISEÑO DE MODELOS PARA EL ALGORITMO

Los algoritmos son la base fundamental del desarrollo de software, son los que conforman las aplicaciones que a su vez se ensamblan en sistemas de software más complejos; los modelos utilizados en los procesos de desarrollo de software son quizás la pieza más importante de la ingeniería de software. Existen varios modelos para llevar a cabo el desarrollo software. Los modelos están conformados por etapas que son generales a todos los enfoques. Las diferencias están básicamente en los tiempos en los cuales se realizan dichas etapas, la simultaneidad, la prioridad que se le da a cada etapa, entre otros elementos. Otras características se pueden observar en detalle en la definición de cada uno de los modelos; estos modelos se facilitan bajo principios como, la creación de algoritmos con código mantenible, escalable y reusable, que sea independiente del lenguaje de programación, que se utilice de forma distribuida en componentes, cada uno con procesos independientes, que presente alta cohesión y bajo acoplamiento. Estas características pueden generar que una aplicación que dé solución a un problema pueda ser usado en diferentes dispositivos y desarrollado en lenguajes de programación diversos.

En el caso de estudio de esta investigación e indagando en la variedad de modelos existentes, utilizados para el diseño de software, se llega a la conclusión de utilizar dos tipos de vistas, que permitirán crear los planos más eficientes para la etapa de desarrollo del algoritmo de Huffman modificado; esto con la intención de ensamblar el componente algorítmico de la solución con la parte de estructura de procesamiento como lo es la red neuronal FFBP. Estas vistas son, el modelo estructural y el modelo dinámico.

3.2.1 Modelo Estructural

El modelado estructural sirve para describir los diferentes tipos y relaciones estáticas existentes entre los diferentes objetos del sistema, a la hora de desarrollar software, se debe modelar correctamente el sistema previamente al desarrollo, pudiendo incorporar el patrón o los patrones de diseño que ayuden a optimizar la solución a un determinado

problema. La principal forma de emplear el modelado estructural es mediante el uso del diagrama de componentes arquitecturales, cuando el sistema es robusto y requiere un alto nivel de abstracción en su despliegue; también mediante un diagrama de clases cuando se trata de un sistema que no requiere un despliegue de componentes mediados por una red de datos. Para modelar la estructura del algoritmo se utilizará un diagrama de clases que haga la abstracción y la especificidad de los elementos que componen el algoritmo y los subprocesos utilizando el modelado matemático que estructura la construcción del grafo, que a nivel de software será el TDA.

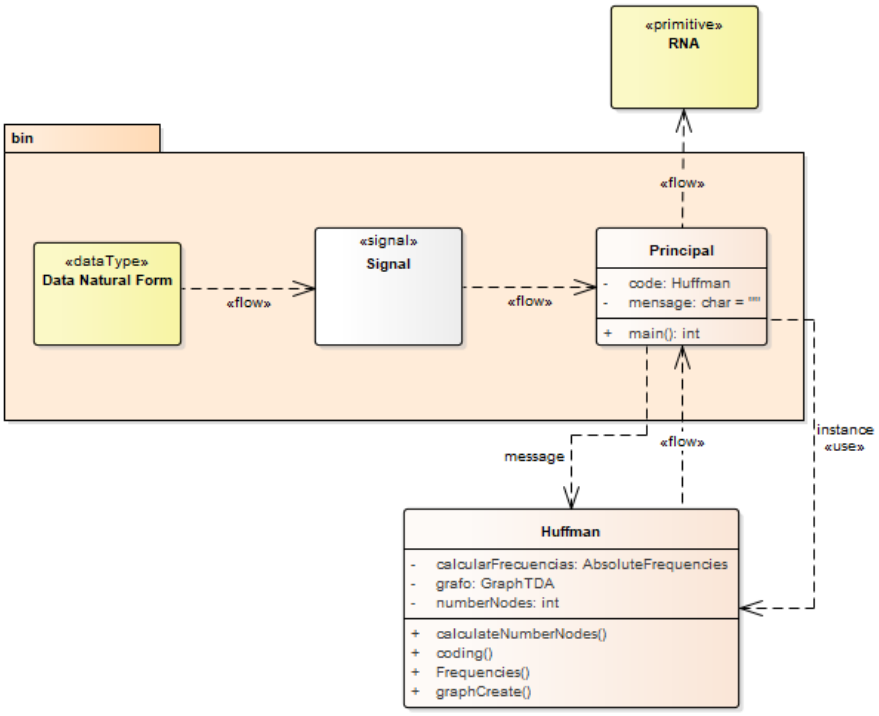
3.2.1.1 Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Los patrones de diseño se clasifican en tres tipos diferentes dependiendo del tipo de problema que resuelven. Estos pueden ser creacionales, estructurales y de comportamiento.

Para el desarrollo de la investigación se utilizaron diferentes tipos de patrones en el diseño estructural, para hacer el modelado de la modificación del algoritmo de Huffman, a continuación, se visualiza y explica su funcionamiento.

3.2.1.2 Patrón Estructural Facade

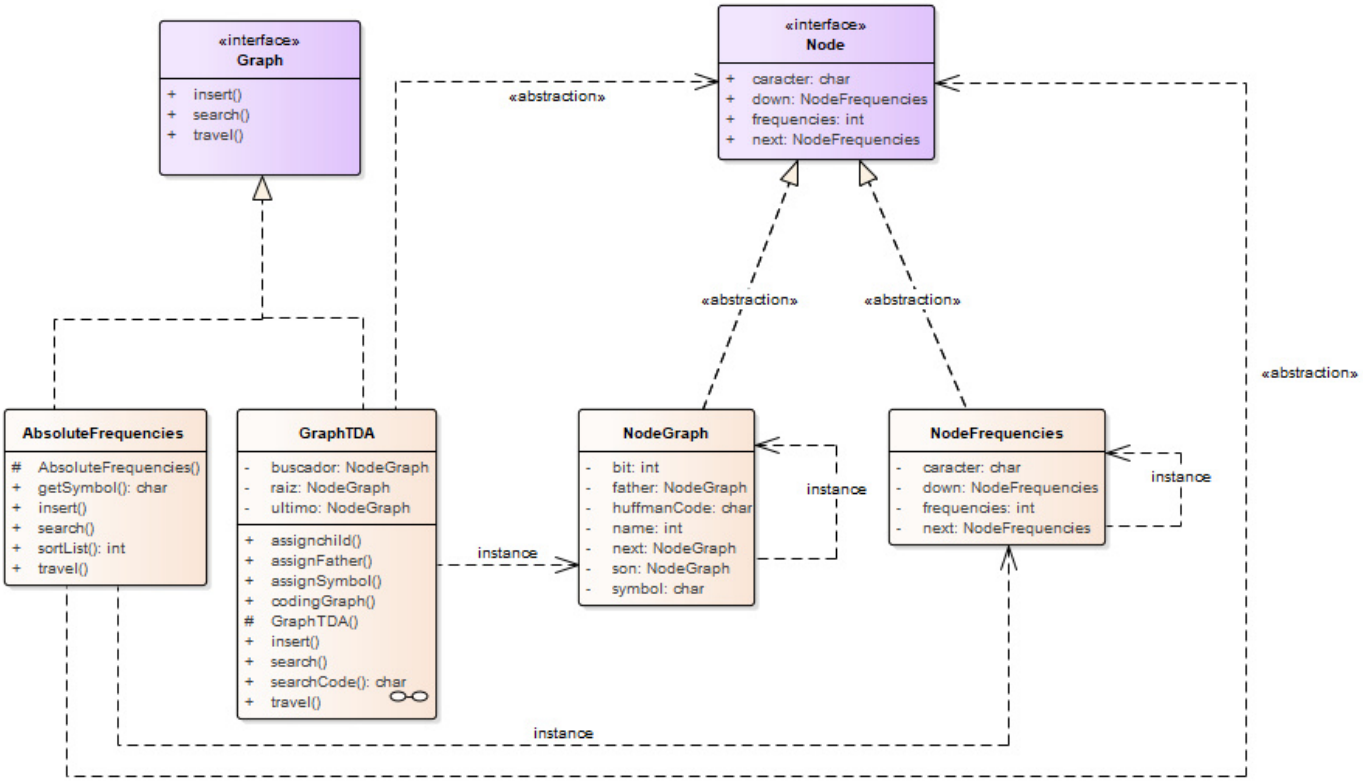
Los patrones estructurales son muy descriptivos, se ocupa de resolver problemas sobre la estructura de los componentes de la aplicación; además estos patrones describen qué relaciones existen entre los componentes del sistema, reduciendo así el acoplamiento.



Gráfica 13 Patrón Estructural Facade

Una Facade (fachada) es un componente que crea una interfaz simplificada para tratar con otra parte del código más compleja, de tal forma que simplifica y aísla su uso; en este caso se utilizará el patrón Facade para gestionar el paso de la señal de entrada que contiene la información hacia las operaciones estadísticas del algoritmo de Huffman, también se utilizará dentro de la misma estructura una salida de información estadística que tendrá destino a las entradas de la RNA; en la gráfica 13 se muestra la estructura del diseño para el componente de entrada al algoritmo mediante un patrón Facade.

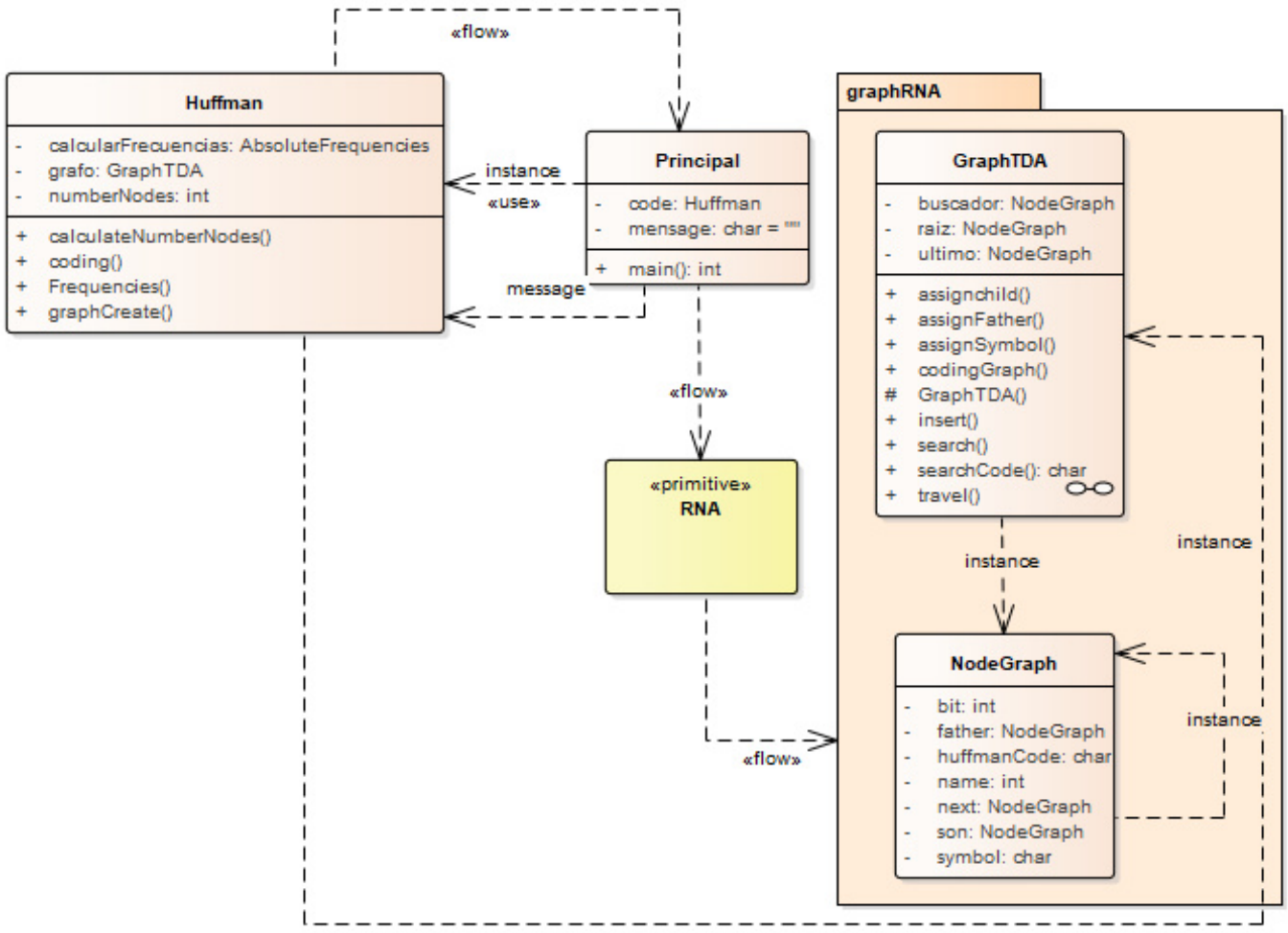
3.2.1.3 Patrón Creacional Abstract Factory



Gráfica 14 Patrón Creacional Abstract Factory

Los patrones creacionales facilitan la tarea de creación de nuevos componentes encapsulando el proceso de creación y diversificando la cantidad de objetos que puede crear; en la aplicación propuesta este patrón se utiliza para crear los componentes de la estructura del grafo de Huffman que darán forma a la topología de la RNA; la ventaja que provee este patrón, radica en que se pueden crear elementos de almacenamiento similares, que conforman el grafo (nodos) y sus relaciones (aristas), cada uno de ellos similar pero con procesos diferentes; por esto último, se denota la importancia de usar la fabricación abstracta mediante interfaces de creación de estos elementos, al terminar las iteraciones del algoritmo, definidas por la longitud de la información y el número de símbolos, en la fábrica de componentes se habrá creado el grafo de Huffman.

3.2.1.4 Patrón estructural Bridge



Gráfica 15 Patrón estructural Bridge

Desacopla una abstracción de su implementación, para que las dos puedan evolucionar de forma independiente. El patrón es utilizado para generar un bajo acoplamiento entre la mecánica del algoritmo de Huffman en su comportamiento canónico y la dinámica de la generación del alfabeto de símbolos que es transferida a la estructura de la RNA, de la misma forma se independiza la salida de la RNA con la secuencia del algoritmo de codificación que sería el último paso del proceso de codificación; la ventaja de esta distribución es que la ejecución de los procesos de captura y codificación de la información se mantienen independientes, con lo que se consigue estática en la topología de la RNA pero dinamismo en la generación de la estructura de activación de la RNA y de generación del alfabeto y su estadística.

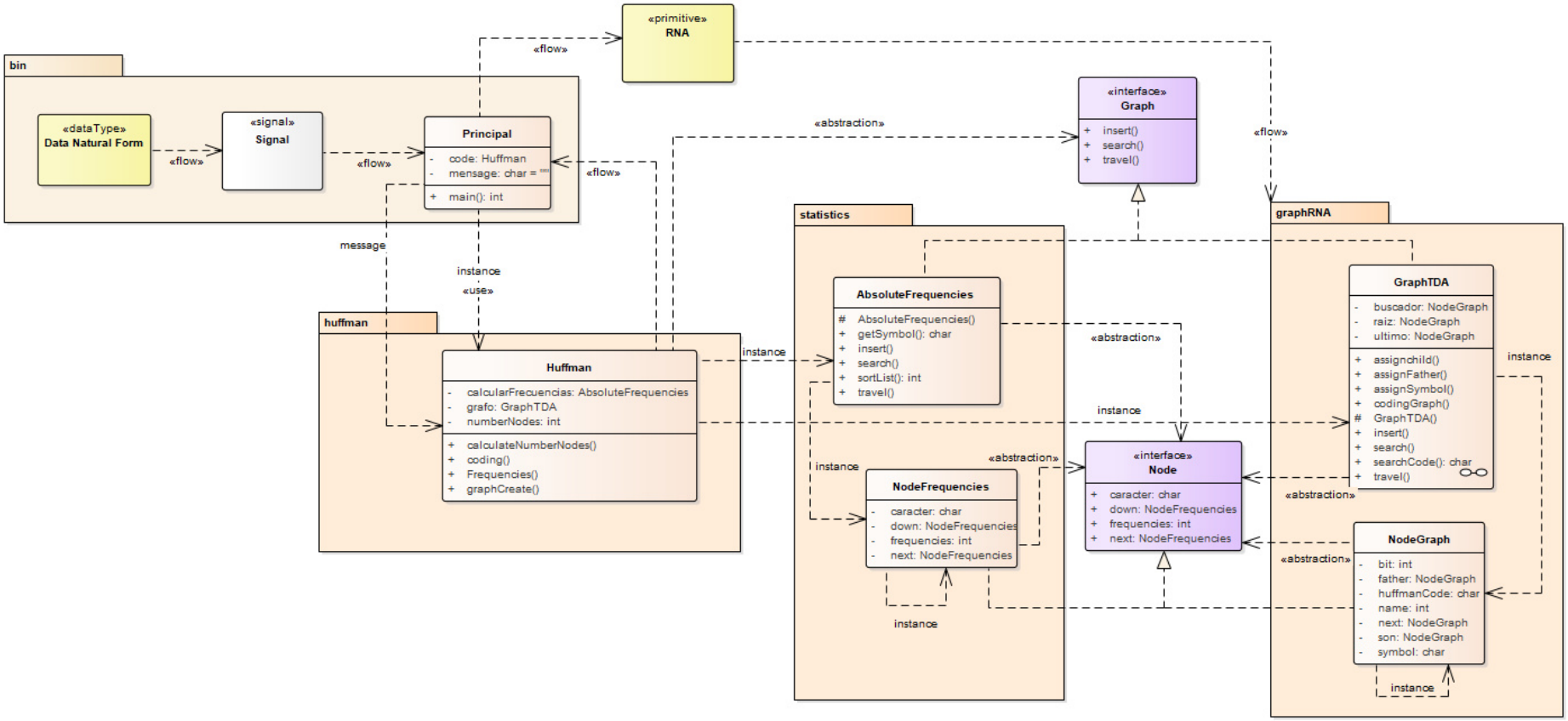
3.2.1.5 Estructura general del Algoritmo

En la siguiente grafica (16), se muestra el ensamblaje general de la estructura de la aplicación que utiliza como base el algoritmo de Huffman, haciendo modificaciones para crear una estructura de tipo grafo y no la de un árbol binario.

La secuencia de relaciones que se observa, empieza con el patrón Facade que separa y desacopla los procesos de entrada y captura de la información de los procesos de entrada y métodos estadísticos del proceso de Huffman; la siguiente etapa es la implementación del patrón Abstract Factory, con el fin de construir la estructura TDA del grafo generado a partir de los procesos estadísticos propios del algoritmo de Huffman; en esta parte es abstracta la creación de componentes según sus características, en una parte de la fábrica se crearán los elementos de almacenamiento y procesamiento o nodos y en la otra fábrica se crearán las conexiones entre nodos o aristas; en términos de implementación estos componentes serán los mismos con procesos de creación diferentes y solo variando pocos atributos, el patrón utilizado para la construcción de la estructura garantiza que esta se genere independientemente de la longitud y naturaleza de la información, también garantiza que la salida que tendrá que utilizarse como las funciones de activación en la RNA sean correctas para los símbolos de entrada.

Por último, la estructura provee un puente para llevar las señales generadas por la información de entrada a la red neuronal con una estructura estática y solo modificada en sus funciones de activación por medio del grafo de Huffman, producto de la fábrica abstracta; el patrón bridge proporciona los métodos de entrada y salida a la implementación de la RNA, así como la salida de la información codificada al proceso externo del algoritmo.

La estructura general descrita se muestra a continuación.



Gráfica 16 Estructura Arquitectural del algoritmo

3.2.2 Diseño dinámico de comportamiento

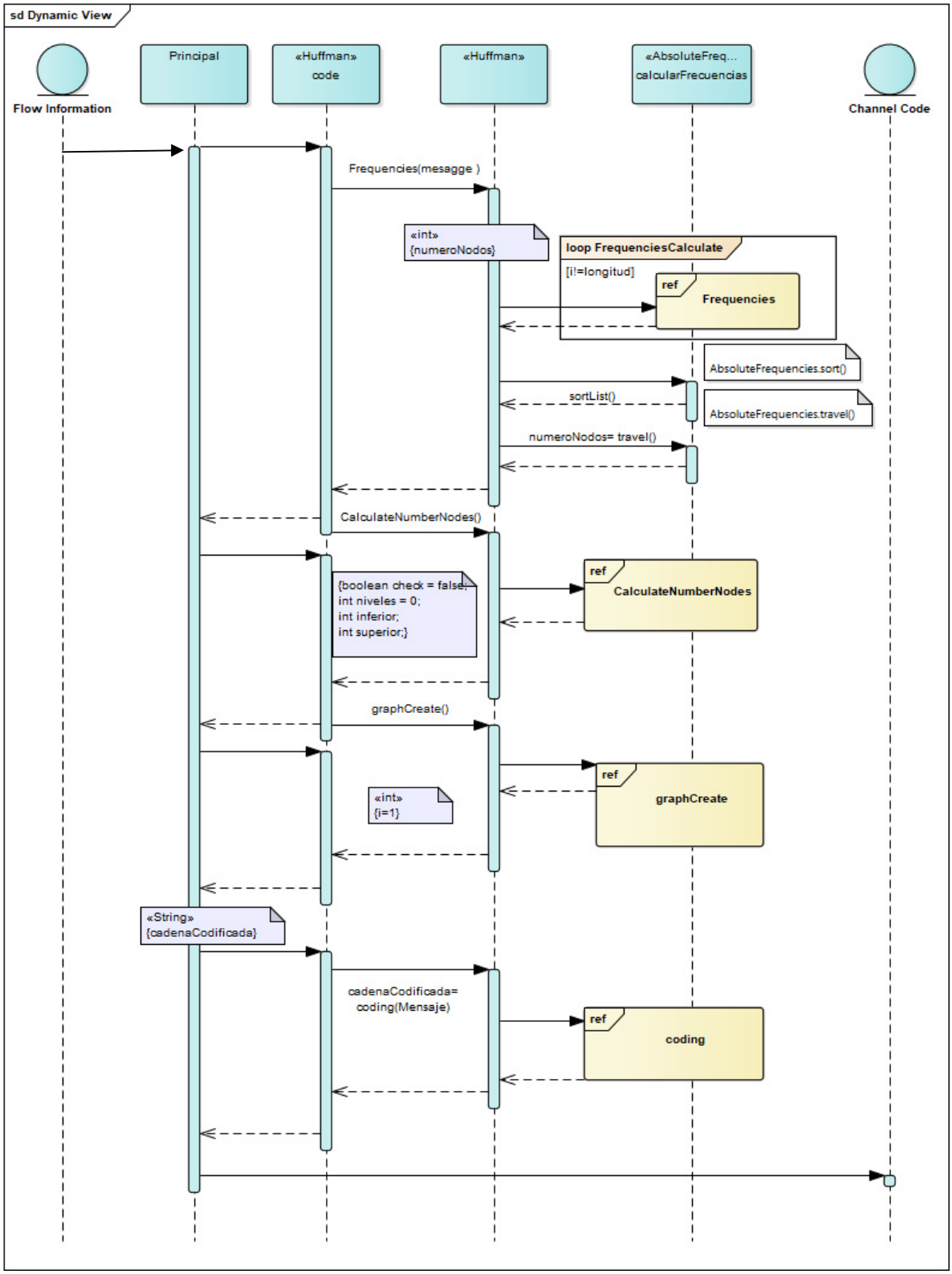
El diseño del modelo dinámico es una parte importante de la concepción de la solución algorítmica de un problema, porque este se centra específicamente en líneas de vida o en los procesos y componentes que coexisten simultáneamente, y los mensajes intercambiados entre ellos para ejecutar las funciones y tareas que lleva a cabo el algoritmo, hasta que este llegue a terminar su ejecución. También mediante este modelo se facilita una vista de factibilidad desde la perspectiva de la implementación ya que lleva al detalle las funcionalidades e instrucciones del algoritmo y de esta manera calcular de manera muy aproximada la complejidad del algoritmo.

En este trabajo de investigación se planteá un diseño dinámico a través de diagramas de secuencias; estos diagramas tienen dos dimensiones, el eje vertical representa el tiempo y el eje horizontal los diferentes componentes, que se relacionan entre sí mediante interacciones de flujo de ejecución representadas por flechas. El tiempo avanza desde la parte superior del diagrama hacia la inferior. Normalmente, en relación con el tiempo solo es importante la secuencia de los mensajes.

Para las necesidades de representación y diseño pertinentes a este trabajo se utilizará la siguiente notación dentro de los diagramas:

Componente, Objeto y línea de vida: Un componente u objeto se representa como una línea vertical discontinua, llamada línea de vida, con un rectángulo de encabezado con el nombre en su interior. También se puede incluir a continuación el nombre del contenedor en orden jerárquico, separando ambos por dos puntos. Si el objeto es creado en el intervalo de tiempo representado en el diagrama, la línea comienza en el punto que representa ese instante y encima se coloca el componente. Si el objeto es destruido durante la interacción que muestra el diagrama, la línea de vida termina en ese punto y se señala con una (X) de ancho equivalente al del foco de control.

La línea de vida de un componente puede desplegarse en dos o más líneas para mostrar los diferentes flujos de mensajes que puede intercambiar un objeto, dependiendo de alguna condición.



Gráfica 17 Diagrama de Comportamiento General

Foco de control o activación: Se representa como un rectángulo delgado superpuesto a la línea de vida del objeto. Su largo dependerá de la duración de la acción. La parte superior del rectángulo indica el inicio de una acción ejecutada por el objeto y la parte inferior su finalización.

Mensaje: Un mensaje se representa como una flecha horizontal entre las líneas de vida de los objetos que intercambian el mensaje. La flecha va desde el objeto que envía el mensaje al que lo recibe. Además, un objeto puede mandarse un mensaje a sí mismo, en este caso la flecha comienza y termina en su línea de vida.

La flecha tiene asociada una etiqueta con el nombre del mensaje y los argumentos necesarios para que se ejecute el proceso al que se está invocando; también pueden ser etiquetados los mensajes con un número de secuencia.

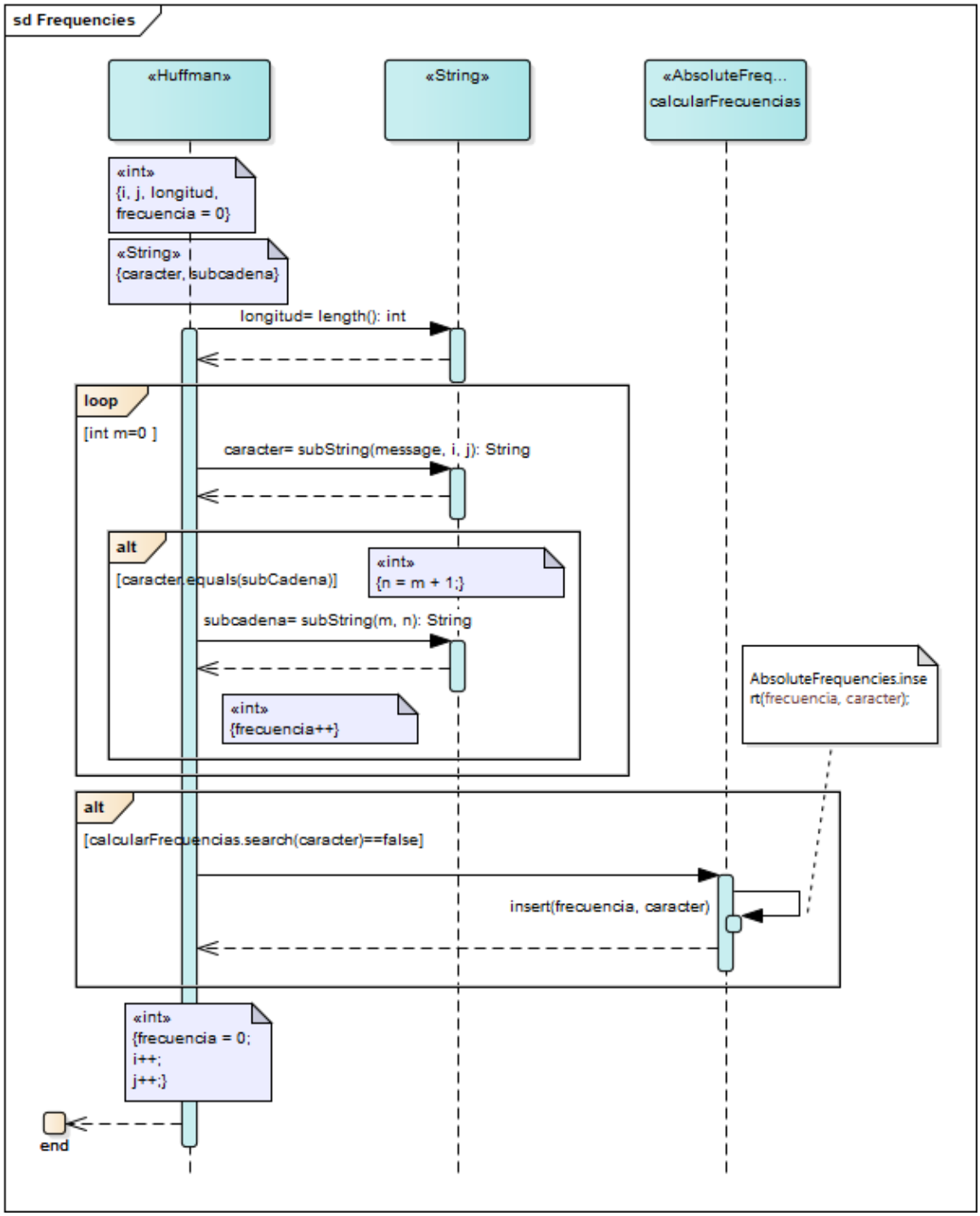
Los mensajes pueden presentar también condiciones e iteraciones. Una condición se representa mediante una expresión booleana encerrada entre corchetes junto a un mensaje y un cuadro de contención el cual llevará dentro los procesos que se ejecutan bajo la condición; una iteración se representa con un asterisco y una expresión entre corchetes, que indica el número de veces que se produce si este no está condicionado y de la misma forma dentro de un cuadro que contiene las instrucciones de la iteración.

En la gráfica 17, se observa el diagrama general de comportamiento del algoritmo de Huffman, con la modificación planteada; allí se observan claramente los pasos, que se asemejan al algoritmo canónico de Huffman con diferencias en la construcción de la estructura TDA y el proceso de codificación; a continuación, se desglosa y explica cada etapa.

3.2.2.1 Componente Estadístico

El algoritmo de Huffman tiene una base matemática fuerte para organizar el alfabeto, además es este proceso estadístico el simple detalle que le da su potencia de compresión, por esta razón la propuesta de algoritmo tendrá un componente estadístico de clasificación similar al algoritmo de Huffman con la diferencia que, los símbolos se organizarán respecto a la frecuencia absoluta de aparición dentro del mensaje a codificar, de esta manera se puede manejar intervalos dentro de la cadena, esperando una mejora en la distribución del

mensaje código y así mejorar la tasa de compresión. Con esta premisa se diseña el siguiente proceso dentro del algoritmo, para clasificar los símbolos y sus agrupaciones basados en la frecuencia absoluta.



Gráfica 18 Diagrama de Comportamiento Paquete del proceso estadístico

En este diseño se tienen en cuenta variables como: los índices que recorrerán el mensaje (i, j), la longitud total del mensaje (cantidad de símbolos) y la frecuencia del símbolo o de la agrupación de símbolos; estas agrupaciones se irán almacenando de manera recurrente

en una subcadena de símbolos que se actualizará por cada iteración. Las iteraciones dependen directamente de la longitud del mensaje de entrada generado por la fuente.

Cálculo de la complejidad

A partir del diagrama de comportamiento dinámico del algoritmo en general o de los subprocesos que lo componen se puede calcular la complejidad algorítmica; a continuación, se presenta la tabla de cálculo de la complejidad del proceso estadístico del algoritmo propuesto.

Tabla 1. Calculo Complejidad Proceso Estadístico

Operaciones Elementales (OE)	17
Condicionales	2
Ciclos	2
Llamados Recurrentes	3

Así para este proceso se puede establecer la complejidad temporal

$$T(n) = 3 + \sum_{i=1}^{n-1} \left(3 + 2 + \left[\sum_{i=1}^{n-1} (2+7)i \right] + 3 \right) \quad (3.7\ a)$$

Donde n, es la cantidad de símbolos del alfabeto del mensaje de entrada; la anterior complejidad hace referencia al peor de los casos, ya que se toma a $n-1$ como el número de iteraciones de los dos ciclos; de aquí en adelante se considerara el caso medio, ya que el peor de los casos calcularía la complejidad de algoritmo en la peor de las situaciones de longitud del mensaje, el cual se puede controlar desde la fuente o de la entrada del algoritmo. El mejor de los casos calcularía la complejidad en una longitud mínima del mensaje lo que no sería una evaluación real para medir el desempeño. Por lo tanto, el caso medio será un promedio del mejor y el peor. En el caso medio, se efectúan la asignación y las operaciones básicas, como unidades elementales en el cálculo, los ciclos son producto de sumatorias de la cantidad de veces que se itera, el número de instrucciones dentro del bucle; por lo tanto, el caso medio de la complejidad algorítmica para el subproceso será:

$$T(n) = 3 + \sum_{i=1}^{\frac{n-1}{2}} \left(3 + 2 + \left[\sum_{i=1}^{\frac{n-1}{2}} (2+7)i \right] + 3 \right) \quad (3.7\ a)$$

Simplificando y aplicando las propiedades de las sumatorias

$$T(n) = 3 + \sum_{i=1}^{\frac{n-1}{2}} \left(5 + \left[\sum_{i=1}^{\frac{n-1}{2}} (9i) \right] + 3 \right) \quad (3.7\ b)$$

$$T(n) = 3 + \sum_{i=1}^{\frac{n-1}{2}} \left(5 + \left[9 \sum_{i=1}^{\frac{n-1}{2}} i \right] + 3 \right) \quad (3.7\ c)$$

$$T(n) = 3 + \sum_{i=1}^{\frac{n-1}{2}} \left(8 + \left[9 \sum_{i=1}^{\frac{n-1}{2}} i \right] \right) \quad (3.7\ d)$$

$$T(n) = 3 + \sum_{i=1}^{\frac{n-1}{2}} \left(8 + \left[9 \left(\frac{\frac{n-1}{2} \left(\frac{n-1}{2} - 1 \right)}{2} \right] \right) \right) \quad (3.7\ e)$$

$$T(n) = 3 + \sum_{i=1}^{\frac{n-1}{2}} 8 + \left(\frac{9}{2} \frac{n^2 - 4n + 3}{4} \right) \quad (3.7\ f)$$

$$T(n) = 3 + \sum_{i=1}^{\frac{n-1}{2}} 8 + \left(\frac{9n^2 - 32n + 27}{8} \right) \quad (3.7\ g)$$

$$T(n) = 3 + \sum_{i=1}^{\frac{n-1}{2}} \left(\frac{9n^2 - 32n + 91}{8} \right) \quad (3.7\ h)$$

$$T(n) = \frac{3}{8} + \sum_{i=1}^{\frac{n-1}{2}} (9n^2 - 32n + 91) \quad (3.7\ i)$$

$$T(n) = \frac{3}{8} + 9\sum_{i=1}^{\frac{n-1}{2}} n^2 + 32\sum_{i=1}^{\frac{n-1}{2}} n \ + 91 \qquad (3.7\ j)$$

$$T(n) = \frac{3}{8} + 9\sum_{i=1}^{\frac{n-1}{2}} n^2 + 32\sum_{i=1}^{\frac{n-1}{2}} n \ + 91 \qquad (3.7\ k)$$

$$T(n) = \frac{75}{8}\left(\frac{(n-1)^2}{24} + \frac{(n-1)}{12}\right) + 32\left(\frac{n^2-4n+3}{8}\right) + 91 \qquad (3.7\ l)$$

$$T(n) = \frac{75}{8}\left(\frac{8n^2-8n-23}{192}\right) + 4n^2 - 16n + 12 + 91 \qquad (3.7\ m)$$

$$T(n) = \frac{600n^2-600n-1875}{1536} + 4n^2 - 16n + 103 \qquad (3.7\ n)$$

$$T(n) = \frac{600n^2-600n-1875}{1536} + 4n^2 - 16n + 103 \qquad (3.7\ o)$$

$$\boldsymbol{T(n) = 4n^2 - 16n + 10} \qquad (3.7\ p)$$

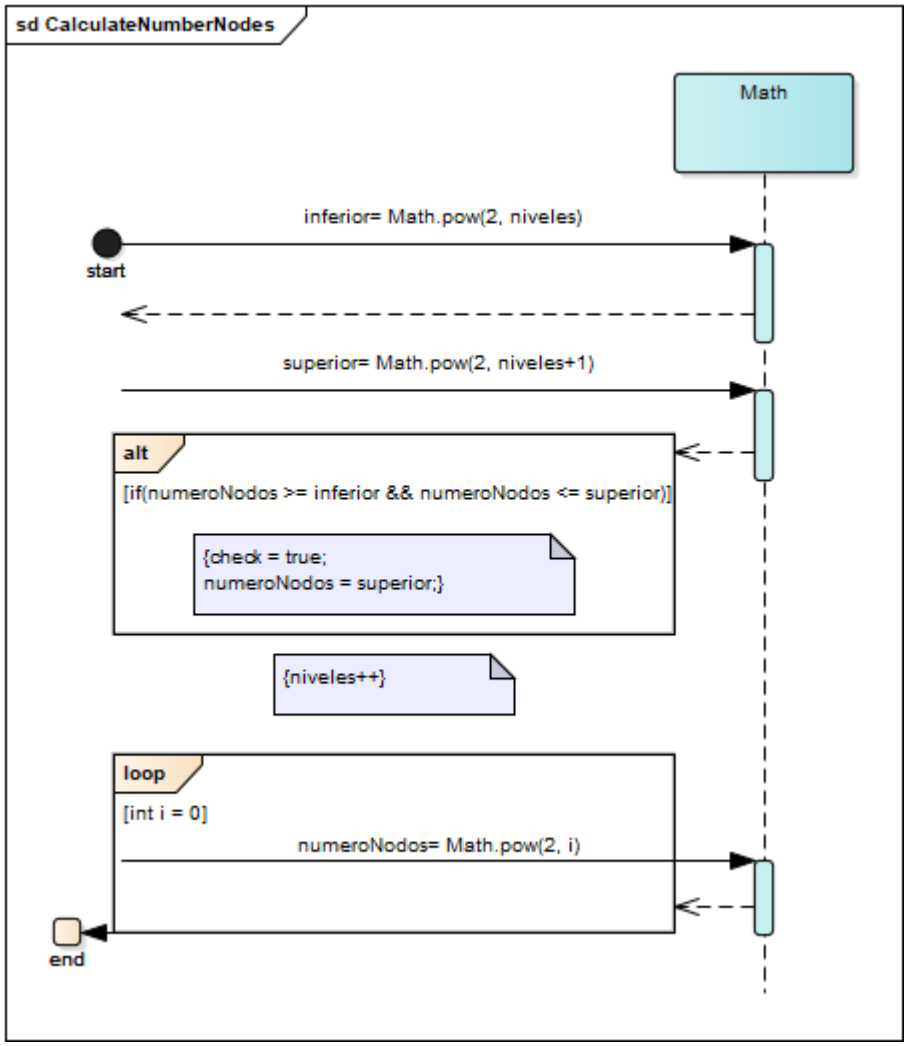
Como se observa al terminar el proceso algebraico se obtiene la complejidad algorítmica del subproceso llevada a cabo por el algoritmo para n símbolos de entrada; lo que se obtiene es que la complejidad es de orden cúbico; puede despreciarse el valor de 10OE y el de tercer término, que es negativo, lo que resta al total, pero también es insignificante con respecto a la suma de los dos primeros, por lo tanto, la complejidad se puede expresar con el orden de:

$$T(n) = 4n^2 + 10 \qquad (3.8)$$

3.2.2.2 Subprocesos Estructurales del grafo Huffman

En el algoritmo propuesto, existen dos subprocesos que corresponden a la construcción del grafo de Huffman que se usará como plano para las funciones de activación de la red neuronal; estos procesos se pueden llamar de estructura del grafo Huffman; a continuación, se describe el diseño comportamental y el cálculo de complejidad de ambos.

3.2.2.2.1 Cálculo de niveles del grafo



Gráfica 19 Diagrama de Comportamiento Calculo de Niveles del Grafo

Este subproceso es el encargado de calcular la cantidad de nodos necesarios para construir el grafo de Huffman, en el resultado se obtiene un valor entero en el cual se encuentran los nodos necesarios para almacenar los símbolos del alfabeto del mensaje, como también los nodos auxiliares necesarios para que el grafo sea un grafo completo; también de este proceso se obtiene el número de niveles que tendrá el grafo, este valor es necesario para determinar el tiempo de respuesta del proceso de codificación a la salida de la RNA.

Cálculo de la complejidad

Tabla 2 Calculo Complejidad Proceso Niveles del Grafo

Operaciones Elementales (OE)	15
Condicionales	1
Ciclos	2
Llamados Recurrentes	3

Así para este proceso se puede establecer la complejidad temporal

$$T(n) = 4 + \sum_{i=1}^{\frac{n-1}{2}} 7i + \sum_{i=1}^{\frac{n-1}{2}} 6i \quad (3.9\ a)$$

En esta y las siguientes expresiones de complejidad no se desglosará todo el proceso matemático, con el fin de no extenderse demasiado en ello; solo se mostrará el planteamiento inicial y el resultado de la simplificación.

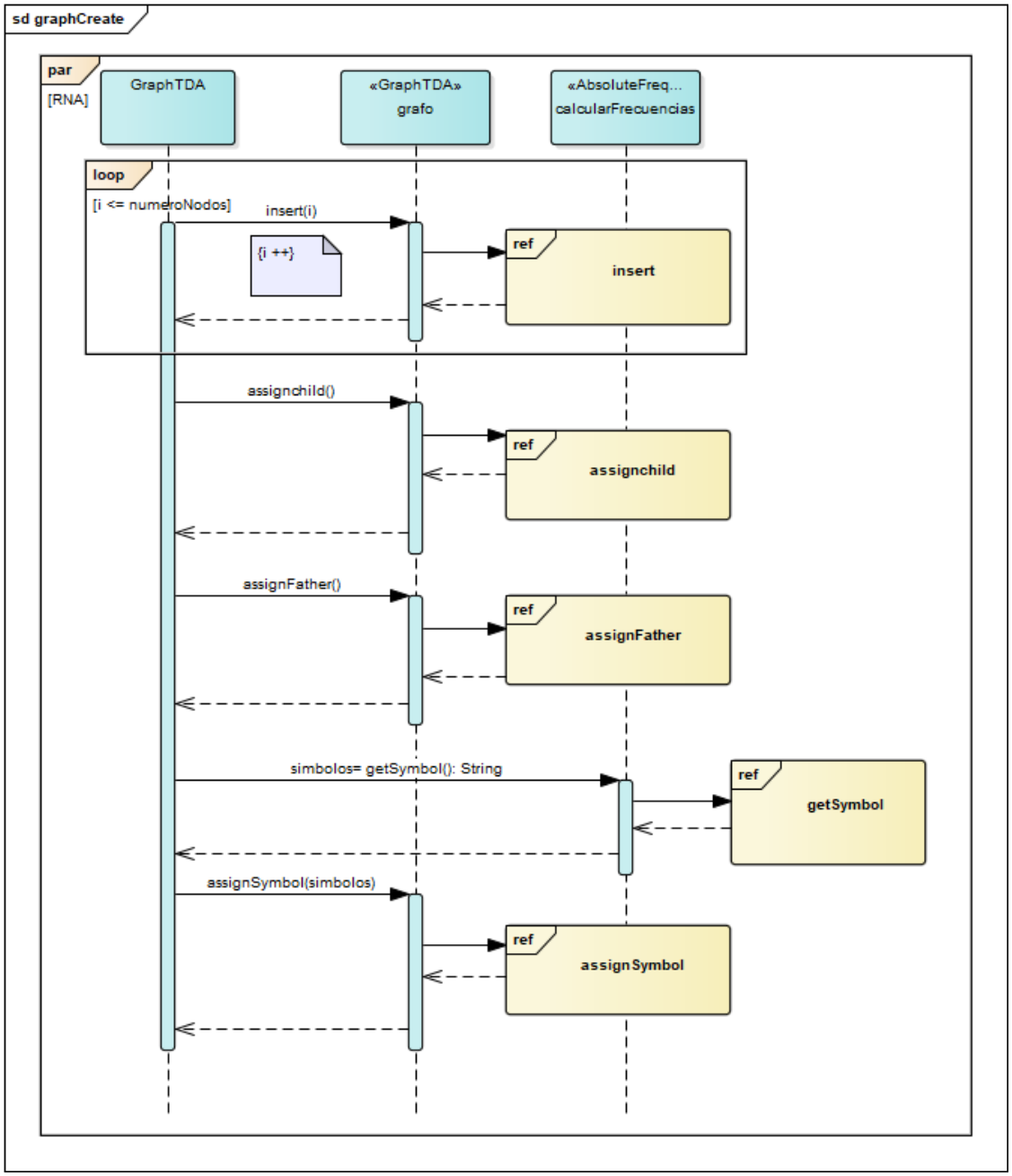
$$T(n) = 4 + \ 2n^2 - 7n - 5 \quad (3.9\ b)$$

$$T(n) = 2n^2 \qquad (3.9\ c)$$

3.2.2.2.2 Proceso de Construcción del Grafo

El algoritmo encargado de construir el grafo de Huffman tiene varios procesos que se encargan de la asignación de la posición de cada uno de los nodos y las distintas conexiones (Aristas), también asigna el símbolo a los nodos que no son auxiliares; cuando el proceso de creación del grafo está completo, la RNA se modificará con base en la estructura del grafo y procesará el mensaje para codificarlo. A la salida de la RNA se obtendrá el código que representa el mensaje; es importante mencionar que en el proceso de decodificación será necesario tener el grafo traspuesto al grafo de Huffman, para definir la RNA del decodificador y así obtener el mensaje original. Por la razón anterior, la construcción del grafo, se define como un grafo dirigido en ambos sentidos, quiere decir esto que un nodo (a) que tiene conexión con un nodo (b), puede establecer un camino entre (a) y (b), pero también entre (b) y (a); esto es conveniente ya que el modelo de RNA utilizado es FF Retro-Propagación (FFBP), y este en un primer paso propaga la señal de entrada hacia adelante y luego hace una propagación hacia atrás para modificar los pesos

sinápticos y polarizaciones, con el fin de reducir el error en la salida. Esto quiere decir que en el decodificador la RNA tendrá las mismas condiciones, con la diferencia que la primera propagación será la que se hace hacia atrás en el codificador y la segunda será la que se realizó hacia adelante, que en el decodificador será la que haga la corrección para mejorar el error.



Gráfica 20 Diagrama de Comportamiento Proceso de Creación del Grafo

Cálculo de la complejidad

Tabla 3 Calculo Complejidad Proceso Creación del Grafo

Operaciones Elementales (OE)	15
Condicionales	1
Ciclos	2
Llamados Recurrentes	4

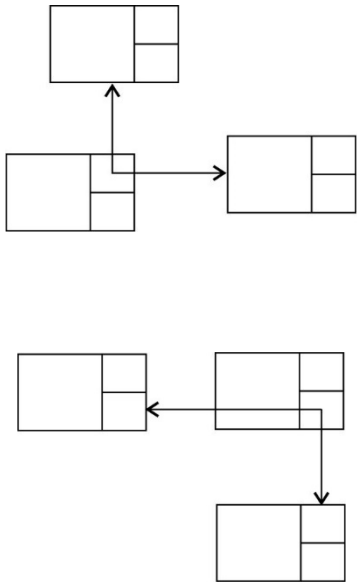
Así para este proceso se puede establecer la complejidad temporal

$$T(n) = 1 + \left(\sum_{i=1}^{\frac{n-1}{2}} (2i + C(n)) \right) + 3C(n) + H(n) \qquad (3.10\ a)$$

$$T(n) = n^2 \qquad (3.10\ b)$$

Donde $C(n)$, es un llamado a las funciones recurrentes de la creación del grafo; estas pueden ser cualquiera de las siguientes tres:

- $C_1(n)$: $Asignación\ conexión\ a\ nodo\ vecino\ o\ hijo\ a\ la\ derecha, arriba, izquierda\ o\ abajo$



Gráfica 21 Asignación de Dirección hacia adelante en el Grafo

- *Cálculo de la complejidad de* $C_1(n)$

Tabla 4 Calculo Complejidad Proceso Propagación hacia adelante

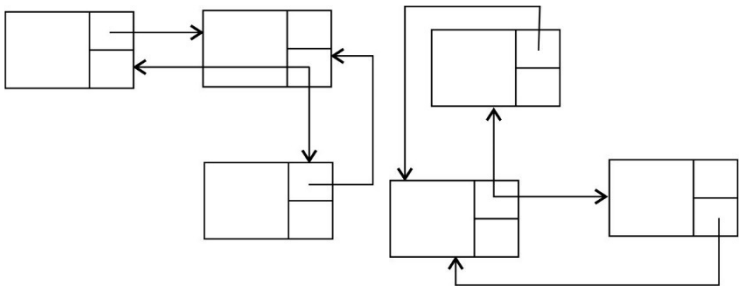
Operaciones Elementales (OE)	8
Condicionales	0
Ciclos	1
Llamados Recurrentes	2

$$T(n) = 3 + \left(\sum_{i=1}^{\frac{n-1}{2}} 5i\right) + \left(\sum_{i=1}^{\frac{n-1}{2}} S(n)i\right) \qquad (3.11\ a)$$

$$T(n) = n^2 + S(n)^2 + 3 \qquad (3.11\ b)$$

Donde $S(n)^2$es el llamado a la función recursiva de búsqueda en profundidad del grafo, con complejidad $T(n) = n^2$.

- $C_2(n)$: $Asignación\ conexión\ a\ nodo\ vecino\ en\ retro\ propagacion$



Gráfica 22 Asignación de Dirección hacia atrás en el Grafo

- *Cálculo de la complejidad de* $C_2(n)$

Tabla 5 Calculo Complejidad Proceso Propagación hacia atrás

Operaciones Elementales (OE)	11
Condicionales	1
Ciclos	1
Llamados Recurrentes	0

$$T(n) = 3 + \left(\sum_{i=1}^{\frac{n-1}{2}} 5i\right) + \max\left(\sum_{i=1}^{\frac{n-1}{2}} 2i\,, \sum_{i=1}^{\frac{n-1}{2}} 3i\right) \quad (3.12\ a)$$

$$T(n) = 2n^2 + 3 \quad (3.12\ b)$$

- $C_3(n)$: $Asignación\ de\ función \frac{y}{o} símbolo\ a\ cada\ nodo$

  - *Cálculo de la complejidad de* $C_3(n)$

Tabla 6 Calculo Complejidad Proceso Asignación de Símbolo

Operaciones Elementales (OE)	11
Condicionales	1
Ciclos	1
Llamados Recurrentes	1

$$T(n) = 4 + \left(\sum_{i=1}^{\frac{n-1}{2}} i\right) + 3 + \sum_{i=1}^{\frac{n-1}{2}} 2i + 1 \quad (3.13\ a)$$

$$T(n) = n^2 + 7 \quad (3.13\ b)$$

- $H(n)$: $Es\ el\ llamado\ al\ proceso\ estadístico\ del\ algoritmo\ para\ obtener\ el\ orden\ de\ los\ símbolos$

  Esta complejidad ya se ha calculado en el apartado 5.3.2.1.

Para terminar el análisis de este proceso se hallará la complejidad total que se requiere para ejecutar la creación del grafo:

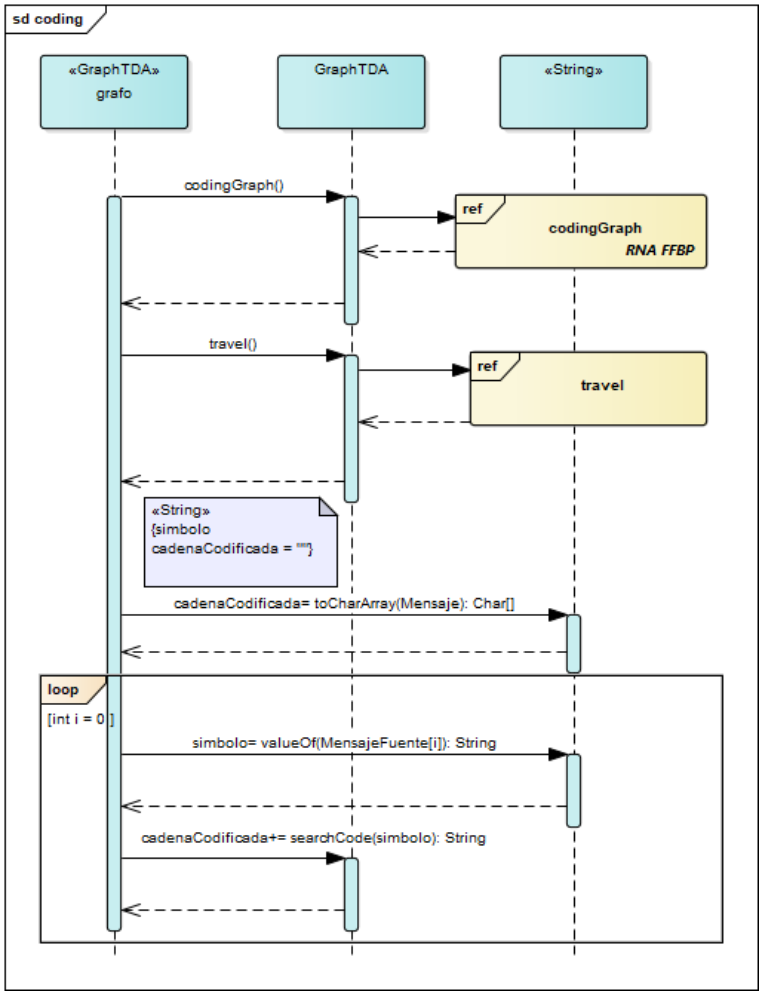
$$T(n) = 1 + \left(\sum_{i=1}^{\frac{n-1}{2}} (2i + C(n))\right) + 3C(n) + H(n) \quad (3.14\ a)$$

$$T(n) = n^2 + n^2 + 7 + 2n^2 + 3 + n^2 \quad (3.14\ b)$$

$$T(n) = 5\,n^2 + 10 \quad (3.14\ c)$$

3.2.2.2.3 Proceso de Codificación

Este es el último proceso de la codificación, cuando el grafo se ha construido y se ha enviado la estructura para modificar las funciones de activación de la RNA FFBP, la salida será la cadena de bits en código Huffman modificado; para esto se debe realizar el recorrido en profundidad sobre el grafo de Huffman; este proceso consta de dos pasos; el primero es la propagación hacia adelante en la red neuronal y el segundo la propagación hacia atrás para ajustar los pesos sinápticos, lo que equivale al proceso de búsqueda de un código en particular que se quiere obtener a la salida de la red dependiendo de la entrada; el nivel de exactitud de este proceso es directamente proporcional al entrenamiento que ha sido aplicado en la red.



Gráfica 23 Diagrama de Comportamiento Proceso de Codificación

Cálculo de la complejidad

Tabla 7 Calculo Complejidad Proceso Codificación

Operaciones Elementales (OE)	9
Condicionales	0
Ciclos	1
Llamados Recurrentes	3

$$T(n) = 6 + \sum_{i=1}^{\frac{n-1}{2}} 5\, i \quad (3.15\, a)$$

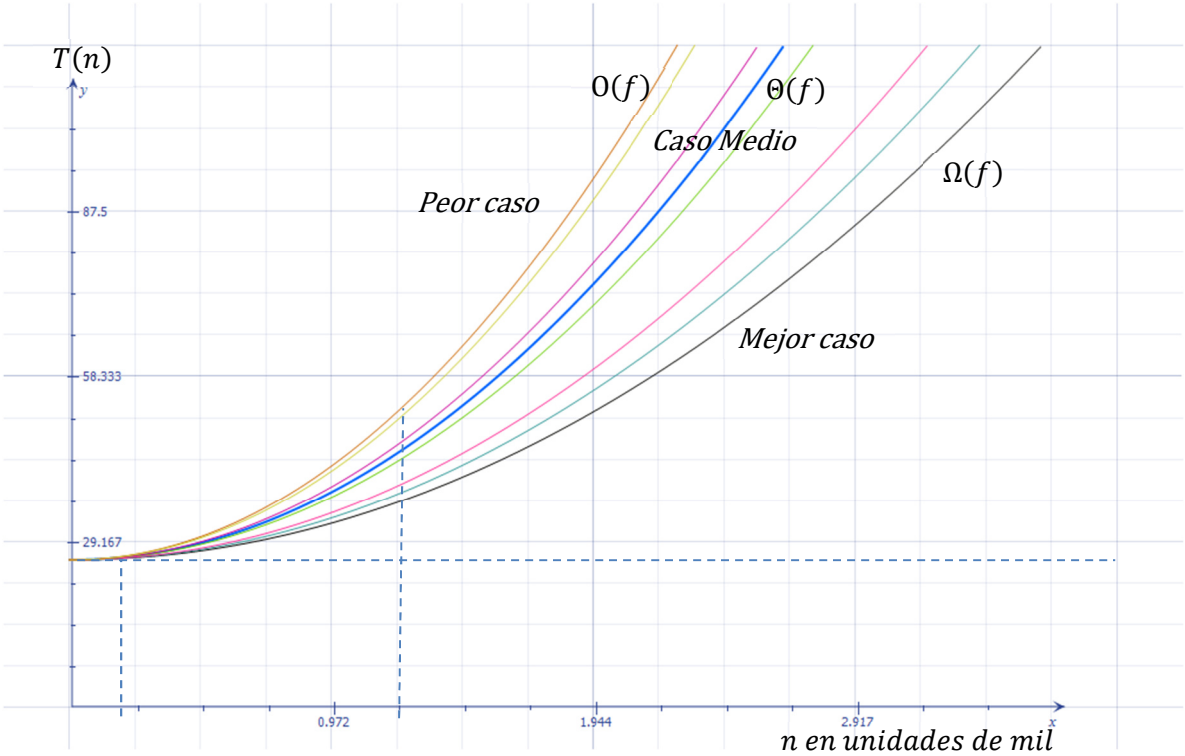
$$T(n) = n^2 + 6 \quad (3.15\, b)$$

3.2.2.3 Complejidad Computacional Total

Con el desarrollo matemático de los cálculos individuales de cada proceso, que conforman el algoritmo de codificación propuesto, puede hallarse la complejidad total y las cotas, para determinar el desempeño en tiempo de ejecución.

$$T(n) = 4n^2 + 10 + 2n^2 + n^2 + 5\, n^2 + 10 + n^2 + 6 \quad (3.16\, a)$$

$$T(n) = 13n^2 + 26 \quad (3.16\, b)$$



Gráfica 24 Cotas de Complejidad del Algoritmo

La gráfica anterior (24) muestra las familias de funciones asociadas a la complejidad algorítmica del procedimiento propuesto; cómo se puede observar en las complejidades individuales de cada proceso, estas son de orden cuadrático; lo que implica que la complejidad sea directamente proporcional al cuadrado de la cantidad de símbolos del alfabeto de entrada n; para el caso de estudio, el desarrollo matemático consideró el caso medio, por tanto la cota adoptada para la proyección de unidades de tiempo de ejecución será la familia$\Theta(f)$ con descripción:

$$\Theta(f) = O(f) \cap \Omega(f) \quad (3.17\ a)$$

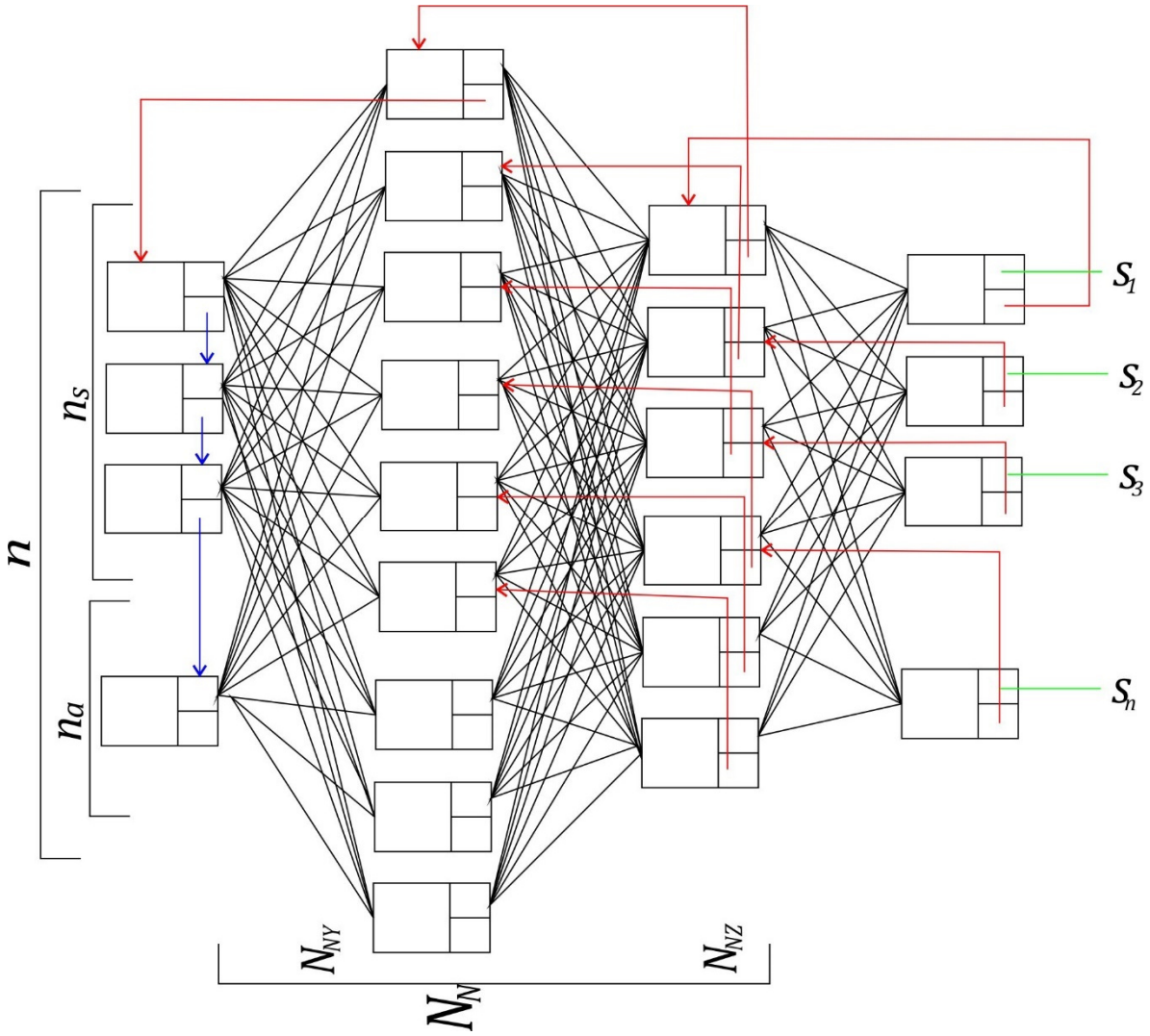
$$\Theta(f) = \{g: n \to [0,\infty) / \exists\, c,d \in R, c,d > 0, \quad \exists\, n_0 \in N \cdot c\, f(n) \leq g(n) \leq d\, f(n), \quad \forall\, n \geq n_0\} \quad (3.17\ b)$$

Intuitivamente, $t \in \Theta(f)$ indica que t está acotada tanto en la parte superior como en la inferior por múltiplos de f, es decir, que t y f crecen de la misma forma y las funciones f consideran todos los casos posibles de valores para n en t_0 e infinito.

Para lograr que la complejidad algorítmica de la solución propuesta, disminuya o se haga independiente del número de símbolos a la entrada, se puede fijar un número de símbolos constante a la entrada seccionando el mensaje de información; también se podría acotar n entre dos valores que hagan que la complejidad llegue a un valor determinado; con esta medida se puede lograr disminuir considerablemente la complejidad disminuyendo la eficiencia del algoritmo en codificación y por ende en compresión. A pesar de esto, puede también disminuir el impacto en la tasa de compresión incrementando la cantidad de ejemplos en la RNA, ya que debe estar más preparada para reconocer patrones dentro de las diferentes cadenas más cortas que pasen por el decodificador.

3.2.3 Diseño de la Red Neuronal FFBP

En la propuesta de modificación del algoritmo de Huffman se utiliza una red neuronal para sustituir la estructura de datos abstracta tipo árbol que usa el algoritmo canónico. Esta RNA, cuenta con neuronas las cuales tienen la función de activación que depende de la estructura del grafo diseñado en los apartados anteriores. Además, el grafo debe ser dirigido y bidireccional. La RNA que cumple con las características de implementación, es la red neuronal Feed Forward Backpropagation, ya que en su algoritmo de entrenamiento permite una propagación en ambos sentidos, tanto hacia adelante como hacia atrás; de esta manera se logra realizar una búsqueda en profundidad controlada sobre todos los niveles del grafo. A continuación, se muestra el diseño de la topología que tiene la RNA FFBP que se utiliza en la implementación de la propuesta.



Gráfica 25 Arquitectura de la RNA Backpropagation para el Algoritmo

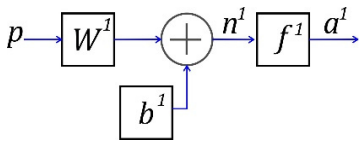
3.2.3.1 Diseño Matemático de la RNA FFBP

En esta sección se planteará la propuesta de modelado matemático de la RNA, con el fin de obtener un diseño claro de las funciones de activación y la forma en la que el algoritmo de entrenamiento recorre la red o propaga las señales de entrada, lo que vendría siendo el procedimiento de búsqueda en profundidad del grafo para realizar la codificación.

Lo primero es considerar que este tipo de redes, son redes multicapa y que se deberá usar una nomenclatura para describir su diseño.

Descripción de las capas de la red

La primera capa de la red se describe de la siguiente forma:



Gráfica 26 Descripción de las capas de la RNA

La ecuación que representa la salida neta de la capa es:

$$a = f(Wp + b) \ (3.18)$$

Donde:

p: $es\ el\ conjunto\ de\ valores\ de\ entrada\ a\ la\ capa, representado\ como\ un\ vector$

W: $es\ el\ conjunto\ de\ pesos\ sinápticos\ que\ inciden\ las\ neuronas\ de\ la\ capa, representado\ como\ una\ matriz\ de\ tamaño\ S\ x\ R, S\ número\ de\ neuronas\ y\ R\ número\ de\ entradas$

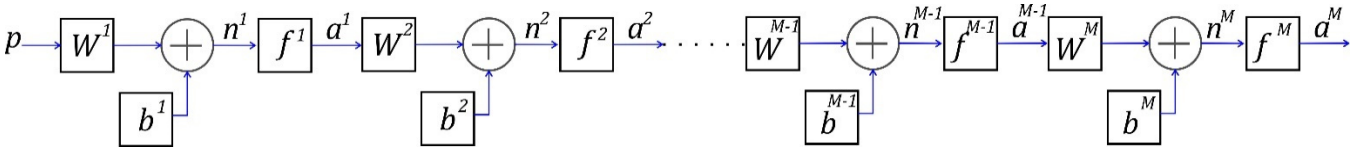
b: $es\ el\ valor\ de\ la\ polarizacion\ de\ las\ neuronas\ de\ la\ capa, representado\ como\ un\ vector$

n: $es\ el\ valor\ de\ la\ entrada\ neta\ a\ la\ funcion\ de\ activación$

f: $es\ la\ función\ de\ activación$

$a\ es\ la\ salida\ neta\ de\ la\ capa$

Cuando se obtiene la salida neta de cada neurona en la capa esta salida pasa a ser la entrada de las neuronas de la siguiente capa; y de la misma forma la salida neta de la segunda capa será la entrada de las neuronas de la tercera y así sucesivamente hasta la capa M, que producirá la salida de la red; a continuación, se muestra el diseño de la red de forma recurrente para M capas.



Gráfica 27 Descripción de la Topología de la RNA

Las dos gráficas anteriores (26, 27) difieren una de la otra en que, la primera es una red monocapa y la segunda ya está compuesta por M capas; el super índice que tienen los componentes de la red indica el número de la capa en la que está ubicado; con este modelado gráfico se pasa a desarrollar las ecuaciones que modelan la red.

En la entrada de la red o capa M^0 se tienen q número de patrones de prueba, conformados de la siguiente manera:

$$(p_1, t_1), (p_2, t_2), (p_3, t_3) \dots \dots \dots \dots \dots \left(p_q, t_q\right) \quad (3.19)$$

La representación matemática de la salida neta de la red multicapa será una ecuación recursiva, la cual es la representación del anidamiento de las ecuaciones de cada una de las capas.

$$a^M = f^M(W^M f^{M-1}(W^{M-1} \dots \dots \dots . . f^2(W^2 f^1(W^1 p + b^1) + b^2) + \dots \dots \dots . . b^{M-1}) + b^M \quad (3.20)$$

Reduciendo la ecuación a su forma recursiva

$$a^m = f^m(W^m a^{m-1} + b^m) \quad (3.21)$$

Proceso de cálculo de los pesos sinápticos y la polarización

Los pesos sinápticos se calculan y se modifican neurona a neurona en cada capa de la red, por lo tanto, cada conjunto de pesos se puede agrupar en una matriz X que representa los valores para cada capa, incluidos los valores de la polarización. Entonces si la red tiene m capas, la representación sería:

$$p_q \rightarrow X^1 X^2 X^3 \dots \dots \dots X^{M-1} X^M \rightarrow a_q^m \quad (3.22)$$

Donde p_q es la información de entrada a la red, $X^{\#\, capa}$ es la matriz de pesos y polarización de una capa y a_q^m es la salida de la red.

Ahora la salida de la red a_q^m, debe ser comparada con la respuesta esperada t_q y de esta manera determinar el error e_q cuadrático para este conjunto de valores de entrada y salida.

$$e_q = t_q - a_q^m = e_q^t e_q \quad (3.23)$$

Ahora se define la función del error cuadrático para poder minimizar su valor, esta función objetivo depende de las M matrices X.

$$f(X^1\ X^2\ X^3 \dots\dots\dots X^{M-1}\ X^M) = e_q^t\ e_q \quad (3.24)$$

Donde

$$X^m = [W^m\ \ b^m]^T \quad (3.25)$$

Lo que quiere decir es que cada matriz X está compuesta por la matriz W de pesos sinápticos de las neuronas de la capa y del vector b de polarizaciones.

Método de Descenso por Gradiente

El método de descenso por gradiente es uno de los algoritmos de optimización más populares en aprendizaje automático, particularmente por su uso extensivo en el campo de las redes neuronales. Este es un método general de minimización para cualquier función f, el algoritmo es especialmente eficiente para espacios bidimensionales como el del presente caso que maneja matrices de tamaño nxm.

Con el método de descenso por gradiente se usará para calcular los pesos sinápticos y las polarizaciones que minimicen el error cuadrático de cada ejemplo durante el entrenamiento de la red.

$$X^m = X^m - \alpha \frac{df}{d\ X^m}\Big|_{X^m - X_0^m} \quad (3.26)$$

El procedimiento se basa en calcular el gradiente de la función objetivo con respecto a cada una de las matrices que contienen los pesos sinápticos; es así como el gradiente se puede representar como una matriz jacobiana conformada por las derivadas parciales de primer orden de la función objetivo con respecto a la salida neta en cada neurona.

$$\frac{df}{d\ X^m} = \left[\frac{df}{d\ x_1^m}\ \ \frac{df}{d\ x_2^m} \dots\dots\dots \frac{df}{d\ x_s^m}\right]^m \quad (3.27)$$

Cada elemento de la matriz es la variación del error cuadrático medio con respecto a los pesos sinápticos y la polarización de cada neurona de cada una de las capas.

Es así como se debe entonces calcular el gradiente $\frac{df}{d\,x_i^m}$ para cada neurona i en la capa m; para ello se descompone la derivada mediante la regla de la cadena, quedando representado como la variación de la función de activación con respecto a la entrada neta n, por la variación de la entrada neta con respecto al vector de pesos sinápticos y polarización en cada neurona x_i.

$$\frac{df}{d\,x_i^m} = \frac{dn_i}{dx_i}\frac{dF}{n_i} \qquad (3.28)$$

Como la entrada neta para la neurona está dada por la expresión $n_i = x_i^m\, z$ y derivando esta expresión con respecto a x_i^m se obtiene que:

$$\frac{dn_i}{dx_i}(x_i^m\, z) = z \qquad (3.29)$$

Y la derivada representada por $\frac{dF}{n_i}$, que es la variación de la función de activación con respecto a cualquier entrada neta, se denominara la sensibilidad de las neuronas denotada por la letra s_i; entonces

$$\frac{df}{d\,X^m} = \left[\frac{df}{d\,x_1^m}\ \ \frac{df}{d\,x_2^m} \ldots\ldots\ldots \frac{df}{d\,x_s^m}\right]^m = [z\,s_1\ \ z\,s_2 \ldots\ldots\ldots z\,s_s]^m \quad (3.29\,a)$$

$$\frac{df}{d\,X^m} = z[s_1\,s_2 \ldots\ldots\ldots\ s_s]^m \quad (3.29\,b)$$

$$\frac{df}{d\,X^m} = z\,s_i{}^m \quad (3.29\,c)$$

$$donde\ s_i{}^m = [s_1\,s_2 \ldots\ldots\ldots\ s_s]^m \quad (3.29\,d)$$

Como z es el vector de las entradas a la neurona aumentado con un (1) de la polarización, puede representarse como:

$$z = \begin{bmatrix} a^{m-1} \\ 1 \end{bmatrix} \quad (3.30)$$

Las entradas a todas las neuronas de la capa m son el conjunto de salidas de la capa anterior, capa $m-1$, sustituyendo en la ecuación del gradiente de la capa, se obtiene:

$$\frac{df}{d\,X^m} = \begin{bmatrix} a^{m-1} \\ 1 \end{bmatrix} {s_i}^m \quad (3.31\ a)$$

$$\frac{df}{d\,X^m} = \begin{bmatrix} a^{m-1}{s_i}^m \\ {s_i}^m \end{bmatrix} \quad (3.31\ b)$$

Para garantizar que las capas queden organizadas en forma vertical (columnas) y las neuronas en forma horizontal (filas), se transpone la matriz que contiene los gradientes.

$$\frac{df}{d\,X^m}^{T} = [{s_i}^m a^{m-1} \quad {s_i}^m] \quad (3.32\ a)$$

Ahora se sustituye la expresión en la ecuación del descenso del gradiente para obtener la ecuación recursiva:

$$X^m = X^m - \alpha \frac{df}{d\,X^m}\Big|_{X^m - X_0^m} \quad (3.33\ a)$$

$$X^m = [W^m \;\; b^m] - \alpha[{s_i}^m a^{m-1} \quad {s_i}^m] \quad (3.33\ b)$$

$$X^m = [W^m \;\; b^m] - [\alpha{s_i}^m a^{m-1} \quad \alpha{s_i}^m] \quad (3.33\ c)$$

$$[W^m \;\; b^m] = [W^m \;\; b^m] - [\alpha{s_i}^m a^{m-1} \quad \alpha{s_i}^m] \quad (3.33\ d)$$

Como la expresión contiene matrices del mismo tamaño (ixm) donde $i\ (filas)$ son las neuronas y $m\ (columnas)$ son las capas de la red; puede descomponerse la expresión en dos partes, la primera para actualizar los pesos sinápticos y la segunda para actualizar las polarizaciones.

$$W^m = W^m - \alpha{s_i}^m a^{m-1} \quad (3.34\ a)$$

$$b^m = b^m - \alpha{s_i}^m \quad (3.34\ b)$$

$$\forall\ m \in [1, 2, 3, \ldots\ldots M] \quad (3.34\ c)$$

Cálculo de la sensibilidad s_i

La sensibilidad representa la derivada del error cuadrático respecto a la entrada de cada capa; esta expresión puede escribirse mediante la regla de la cadena; aquí se va a generar una expresión para calcular la sensibilidad de la capa $m-1$, ya que para hallar este valor no se hará una búsqueda o propagación hacia adelante, sino una retro propagación, por lo tanto, se requiere la sensibilidad de la capa m para calcular la sensibilidad de la capa anterior y así sucesivamente.

$$s_i = \frac{dF}{dn^{m-1}} = \frac{dn^m}{dn^{m-1}} \frac{dF}{dn^m} \quad (3.35)$$

Dado que $\frac{dF}{dn^m}$, es la variación de la función de activación con respecto a la entrada neta de la capa m y $\frac{dn^m}{dn^{m-1}}$, es la variación de la entrada neta en la capa m con respecto a la entrada neta de la capa $m-1$, sustituyendo para la notación de las sensibilidades de cada capa se obtiene la expresión recursiva de s.

$$s_i{}^{m-1} = \frac{dn^m}{dn^{m-1}} \; s_i{}^m \quad (3.36)$$

De la que se puede encontrar que la sensibilidad de la capa $m-1$ esta en función de la sensibilidad de la capa posterior. Ahora se describe la entrada neta de la capa m, así:

$$n^m = W^m f^{m-1}(n^{m-1}) + b^m \quad (3.37)$$

Donde la entrada neta de cada capa depende de los pesos sinápticos asociados, de la función de activación de la capa anterior, que a su vez depende de la entrada neta de la capa anterior y de la polarización de las neuronas de la capa.

La variación de la entrada neta de la capa $m-1$ con respecto a la entrada neta de la capa posterior es entonces la derivada parcial de la variación de la función de activación de la capa con respecto a la entrada neta por la entrada neta de la capa posterior con respecto a la función de activación.

$$\frac{dn^m}{d\,n^{m-1}} = \frac{df^{m-1}}{d\,n^{m-1}} \frac{dn^m}{df^{m-1}} \quad (3.38)$$

Sustituyendo en la expresión de sensibilidades

$$s_i{}^{m-1} = \frac{dn^m}{dn^{m-1}}\ s_i{}^m \quad (3.39\ a)$$

$$s_i{}^{m-1} = \frac{df^{m-1}}{d\ n^{m-1}}\frac{dn^m}{df^{m-1}}\ s_i{}^m \quad (3.39\ b)$$

$$s_i{}^{m-1} = \frac{df^{m-1}}{d\ n^{m-1}}\ W^T\ s_i{}^m \quad (3.39\ c)$$

$$\forall\ m\ \in [M, M-1, \ldots\ldots.2]$$

Para hallar la sensibilidad en la última capa, se necesita conocer el valor del error cuadrático y la variación de la función de activación de la última capa con respecto a la entrada neta, es así que:

$$s^m = -2\frac{df^m}{d\ n^m}\ e_q \quad (3.40)$$

Cálculo de la Variación de la función de activación $\frac{df^m}{d\ n^m}$

La función de activación es independiente de cada capa y neurona y esta solo depende de la entrada neta en cada neurona; es por esto que cada neurona tendrá su propia función de activación y esta no depende de las neuronas de la misma capa, como se necesitan las funciones de activación para todas las neuronas de cada capa, esta expresión representa una matriz de la siguiente forma:

$$\frac{df^m}{d\ n^m} = \begin{bmatrix} \frac{df^m{}_1}{d\ n^m{}_1} & \ldots\ldots & \frac{df^m{}_i}{d\ n^m{}_i} \\ \vdots & \ddots & \vdots \\ \frac{df^1{}_1}{d\ n^1{}_1} & \ldots\ldots & \frac{df^1{}_i}{d\ n^1{}_i} \end{bmatrix} \quad (3.41)$$

Como la dependencia entre la función de activación depende estrictamente de la entrada neta de la neurona, se puede simplificar la matriz anterior como una matriz que contiene las funciones de activación que tienen una dependencia directa.

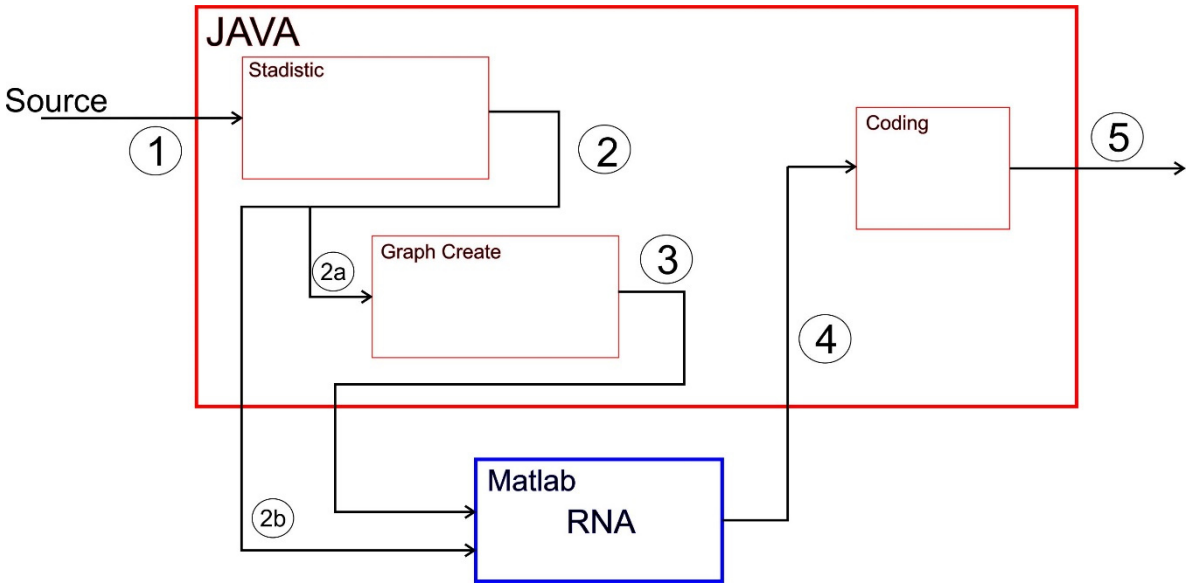
$$\frac{df^m}{d\,n^m} = \begin{bmatrix} \frac{df^m{}_1}{d\,n^m{}_1} & \dots\dots & 0 \\ \vdots & \frac{df^{m-1}{}_2}{d\,n^{m-1}{}_2} & \vdots \\ 0 & \dots\dots & \frac{df^1{}_i}{d\,n^1{}_i} \end{bmatrix} \quad (3.42)$$

Ahora se tiene la matriz que almacena las funciones de activación de cada neurona y su dependencia directa con las demás neuronas, lo que es una analogía de una matriz de adyacencia que representa un grafo.

De esta manera la construcción de la red neuronal mediante el algoritmo Backpropagation, será modelada por el grafo de Huffman mediante las funciones de activación en cada neurona y representada mediante la matriz de funciones a la matriz de adyacencia del grafo.

3.3 IMPLEMENTACION

La fase de implementación del prototipo que describe el proceso que se quiere realizar para la codificación y compresión de la información, se dividió en dos partes; el primer componente fue realizado en lenguaje de programación JAVA, basado en los diseños estructural y dinámico relacionados en el apartado (4.3), la segunda parte fue la implementación de la RNA que se construyó en base al diseño descrito en el apartado (4.3.3) y se utilizó la herramienta Matlab.



Gráfica 28 Diagrama de Bloques Funcional del Prototipo

Estos dos componentes se integraron mediante el recorrido del flujo de información de forma secuencial, de manera que la salida del primer componente es la entrada del segundo quien arroja la salida y es presentada de nuevo en el primer componente; lo que quiere decir que la implementación de la RNA es un elemento externo usado dentro del algoritmo de codificación.

En la gráfica 28, se muestra el diagrama de flujo del codificador; también se enumeran los pasos que se llevan a cabo en el procesamiento de la información.

1. La información entra en su forma natural, hacia el componente encargado de extraer el conjunto de todos los símbolos que representan la información y determinar la frecuencia relativa de cada uno con respecto al total de información a codificar.
2. La salida del bloque de estadísticas es una lista de símbolos, cada uno con una frecuencia relativa asociada que depende de la cantidad de veces que aparece el símbolo dentro de la información; este producto debe ser entregado a dos destinos en instantes de tiempo diferentes. Primero se entrega al bloque de creación del grafo (2a), que se encarga de construir el grafo dirigido K completo, a partir del conjunto de símbolos; el segundo (2b), se hace con un retardo equivalente a la complejidad temporal $T(n)$, del proceso de creación del grafo y es la entrada a la red neuronal.

3. La salida del bloque de creación del grafo es una lista de listas con la información de los nodos y su valor, que representa los símbolos, caminos y su direccionamiento. Esta estructura TDA, es la entrada hacia la RNA, la cual modificara su topología en términos de las funciones de activación con respecto a cada uno de los nodos del grafo; lo que quiere decir que cada neurona modificara los parámetros de la función de activación dependiendo de la información contenida en los nodos del grafo.
4. El bloque RNA tiene dos entradas la primera es la salida del bloque de creación del grafo, esta entrega se hace una sola vez por trama de información y la segunda es la entrada de la información estadística de la trama, que se envía recurrentemente, la cantidad de veces que se divide el total de información en segmentos.
   La primera entrada se utiliza para ajustar los parámetros de la RNA y el segundo se usa para dar entrada a la RNA y obtener el código; por tanto, la salida del bloque RNA serán los segmentos de la información en su forma codificada.
5. La salida del bloque RNA es el código producto del procesamiento de la información, este código esta segmentado, por lo tanto, se ensambla en el bloque coding y se entrega como salida total del decodificador.

3.4 PRUEBAS Y ANALISIS DE RESULTADOS

Se realizaron pruebas sobre el funcionamiento y resultados obtenidos del algoritmo frente a la entrada de información y su código resultante. También se realizó el cálculo del tamaño del código de la información con respecto a la fuente y a una codificación ASCII (acrónimo inglés de American Standard Code for Information Interchange), de esta manera comparar y definir la razón de compresión del algoritmo.

Otro factor evaluado durante las pruebas es la concordancia de la complejidad temporal teórica, previamente calculada y el desempeño computacional experimental del algoritmo. De la comparación de estos dos valores se determinará el indicador de la eficiencia del algoritmo con respecto a los recursos de procesamiento y almacenamiento.

El apartado de resultados se encuentra enfocado a los valores obtenidos del algoritmo de codificación en las pruebas; los parámetros hallados son: número de bits por trama, numero de bytes por trama, tasa o razón de compresión y valor de complejidad computacional.

Trama (t)

Una trama es un segmento de información, que en conjunto componen el total de la misma; como se describirá posteriormente, en este trabajo las pruebas realizadas utilizaron información en forma de videos, por lo tanto, una trama es un segmento de video con duración de un segundo.

Número de bits por trama $\left(\frac{b}{t}\right)$

Este indicador relaciona la cantidad de bits que conforman una trama; que a su vez está compuesta por la información obtenida de un píxel sumada a la información digital obtenida por muestreo de los canales de audio en el instante de tiempo en el que se encuentra dicho píxel en procesamiento. Este valor es estático en la codificación ASCII y su tamaño es de cinco bytes, conformados por tres bytes de información del color del píxel y dos bytes con la información del sonido; para un total de 40 bits.

El valor de numero de bits por trama para la codificación Huffman, es dinámica ya que la longitud en bits para cada trama es diferente y depende de su probabilidad, dentro del subconjunto total de tramas o símbolos que están contenidos dentro del conjunto de fotogramas que componen el video. Por consiguiente algunas de las tramas tendrán longitudes más largas, incluso que de 40 bits o más pequeñas que un byte; entonces este indicador será representado para la codificación Huffman del algoritmo propuesto, como el promedio aritmético de todas las longitudes de las tramas de un fotograma, que a su vez se promediara con las obtenidas de cada uno de los fotogramas del total que componen el video, para un estimado general de las longitudes o número de bits por trama que puede ofrecer el algoritmo en cada video de prueba.

Número de bytes por trama $\left(8\frac{b}{t}\right)$

En concordancia con el parámetro explicado en el párrafo inmediatamente anterior, el número de bytes por trama, es el mismo número de bits por trama agrupado en conjunto de ocho bits para sintetizar y estandarizar la lectura de los resultados; sin embargo, el procedimiento para calcular este parámetro se ejecuta directamente de la salida del codificador y al igual que el número de bits por trama se hace con un contador de bits, lo que garantiza, dos valores comparativos para determinar errores o inconsistentencias tanto en la codificación como en el análisis de resultados.

Tasa o razón de compresión $\left(C_{\frac{H}{A}}\right)$

Este parámetro porcentual, dado entre [0, 1], representa la proporcionalidad entre el valor de la codificación Huffman clásico y el valor encontrado por el algoritmo en la codificación Huffman modificada, de esta manera, si el valor de la tasa de compresión es menor, mejor será el resultado, ya que indica la proporción de bits que utilizo la palabra código Huffman modificado con respecto a Huffman clásico. Con lo anterior se obtendrá que tanto se comprimió la información con respecto a la fuente en su estado original y su representación en bits Huffman.

Diferencia con la tasa de Compresión Teórica $\left(C_{\frac{H}{Sh}}\right)$

El valor de la diferencia de la tasa de compresión teórica de Shannon con la tasa de compresión obtenida con la codificación Huffman modificada se obtiene con la siguiente expresión:

$$\mathrm{C}_{\frac{\mathrm{H}}{\mathrm{Sh}}} = \mathrm{C}_{\frac{\mathrm{H}}{\mathrm{A}}} - I(S) \quad (3.43)$$

De esa forma se encontrará la cantidad porcentual entre [0, ∞], que representa lo que le hace falta al código Huffman modificado para llegar al límite teórico de Shannon.

Complejidad computacional$(\boldsymbol{T(n)})$

Este indicador es el valor numérico, obtenido directamente de la ecuación $T(n) = 13n^2 + 26$, que representan el total de la complejidad hallada para el algoritmo, este valor se evalúa dentro de la cota de complejidad de cada proceso; en la ecuación se sustituye el valor de n, como la cantidad de símbolos en el alfabeto y el cual genera el número de operaciones que debe ejecutar la máquina y cuya magnitud depende directamente del número de veces que se tienen que ejecutar los procesos estadísticos y de creación del grafo; en otras palabras depende directamente de la cantidad de símbolos encontrados para cada fotograma y por consiguiente para cada video.

Numero de iteraciones (It_{RNA})

El número de iteraciones o numero de la iteración representa la cantidad de veces que se ha ingresado la misma trama de información a la entrada de la red neuronal; con este valor

se expresa la cantidad total de veces que se utilizó la red neuronal para codificar una misma trama, también el número de la trama en la cual se encontró el mejor resultado; sin embargo, este número es variable en cada una de las tramas y osciló entre 1 y 12 iteraciones.

3.4.1.1 Características de la fuente de información.

Para realizar las pruebas del algoritmo, se utilizaron ocho archivos de video, esto debido a que este tipo de información es muy utilizada en la actualidad. Un ejemplo de lo anterior son los servicios de streaming, los cuales realizan transmisión de video de forma síncrona desde el emisor hasta el receptor, también las conferencias desde puntos remotos, entre otros. El video también es el tipo de información que ocupa más espacio de almacenamiento y por lo tanto requiere mayores recursos en la transmisión, esto se debe a que no solo está compuesto por imágenes que se reproducen en una unidad de tiempo (fotogramas), sino que además requiere espacio para almacenar canales de audio. Por las razones anteriores, probar el algoritmo de compresión de Huffman modificado utilizando RNA, para procesar este tipo de información, dará más claridad al momento de evaluar su desempeño tanto en la codificación/compresión como en la complejidad computacional esperada.

La información seleccionada para realizar las pruebas, son fragmentos de video; con la misma duración, donde la codificación obtenida del video no depende del formato origen de este, ya que el algoritmo lo desglosa en tramas de un segundo seguido de evaluarlos en frames o fotogramas. Que a su vez son divididos en pixeles, con lo que se construye una matriz de vectores con tamaño (1,3) donde se almacenan los valores enteros entre 0 y 256 que representan el valor de cada píxel en formato de color RGB, para un total de 16 millones de valores posibles para cada pixel. A esto se le suman dos canales de audio (estéreo), cada uno de 22 KHz de ciclo de muestreo, que aumentará en 16 bits el espacio necesario de almacenamiento o transmisión.

Cada archivo de video utilizado tiene las siguientes características:

Tabla 8 Características técnicas de los videos de prueba

Concepto	Valor	Unidad
Duración	30	Segundos
Resolución	1280x720	Píxel x Píxel

Numero de pixeles por Fotograma	921600	Píxel
Velocidad fotograma	24	Frames/segundo
Velocidad pixel	22118400	Pixel/segundo
Bits por píxel	24	bits
Bytes por píxel	3	Bytes
Sonido	2	Canales (Estéreo)
Muestreo de cada Canal de sonido	44	KHz
Muestreo del sonido en bits	16	Bits
Muestreo del sonido en Bytes	2	Bytes
Total, Bytes por trama	5	Bytes

En cuanto a las características audiovisuales propias de cada video, se utilizaron cuatro tipos diferentes.

Tipo de Video (1)

Es una secuencia en la cual los fotogramas cambian frecuentemente y no hay un patrón de similitudes fácilmente reconocible durante un intervalo de tiempo, lo que ocasiona que la cantidad de los símbolos dentro del alfabeto a la entrada del codificador varían en cada grupo de tramas que entran al proceso.

Tipo de Video (2)

es una variación del primero, ya que en la primera mitad tiene las mismas características, pero en la segunda los fotogramas no varían mucho de uno a otro y los patrones de trama a trama que ingresan al codificador no varían mucho en intervalos de tiempo más largos.

Tipo de Video (3)

Tiene las mismas condiciones que el de tipo dos, con la diferencia que en la primera mitad del video los fotogramas no varían mucho de uno al otro en intervalos de tiempo moderados, mientras que en la segunda mitad se comporta como el video de tipo uno. La razón por la cual se editaron los videos de prueba, con estas condiciones, es porque la RNA se comporta de forma distinta para cada caso, debido al entrenamiento recibido, por esta razón en importante analizar y evaluar los resultados en cada caso y así mismo ver como mejora o empeora el desempeño del algoritmo en varias iteraciones utilizando la RNA ya entrenada.

Tipo de Video (4)

El video tipo cuatro; es una secuencia de imágenes y sonido que es similar en el trascurso de la reproducción, lo que quiere decir que los fotogramas no varían mucho de uno a otro;

con lo que se espera que el alfabeto de símbolos no aumente en número, clasificado por la probabilidad de aparición.

3.4.2 Procedimiento de las pruebas

A continuación, se enumera secuencialmente el procedimiento que sigue el algoritmo para la codificación de cada archivo de entrada:

1. Modificar los archivos de video, para que tengan las características mencionadas, con el fin de encontrar los resultados a entradas de información lo más similares posibles y así poder realizar comparaciones.
2. Se carga el archivo de video.
3. El video se fragmenta en tramas de 1 segundo de duración
    a. Cada trama se divide en frames (fotogramas).
4. Cada fotograma se divide en pixeles, creando la matriz de referencia con tres valores enteros para cada píxel.
5. Se recorre cada fotograma, píxel a píxel para recoger los valores de color de cada uno.
    a. Se codifican los valores de cada píxel con la codificación ASCII.
    b. Se concatena el código de cada píxel recorrido con los anteriores.
    c. Se almacenan los valores RGB que corresponden a cada píxel en la matriz de referencia de cada fotograma.
6. Se recorre el total de elementos de la matriz de referencia, pasando los valores almacenados en ella a la entrada del codificador en tramas de 8 bits (1 byte).
7. Se realiza el cálculo teórico Shannon, para obtener la longitud de la palabra código.
8. Se realiza la clasificación estadística de cada trama de 5 bytes con respecto a las frecuencias relativas de aparición de cada trama.
9. Se realiza la codificación con el método clásico de Huffman
10. Creación del grafo con base a la clasificación estadística.
11. Se adaptan los parámetros de la RNA con base al grafo producto de la clasificación estadística de la siguiente manera:
    a. Las neuronas que representan nodos existentes en el grafo adoptaran una función de transferencia sigmoide Logaritmica; mientras que las neuronas

que representen nodos auxiliares usaran una función de activación sigmoide Tangencial.

b. Los pesos sinápticos de las neuronas de la red tendrán un valor constante para todas las neuronas de 0.5, con el fin de que para el primer entrenamiento la RNA represente un grafo dirigido K-Completo, con los mismos ponderados en las aristas, este valor medio garantiza la convergencia más rápida entre el 1 o el 0 binario que vaya a representar la salida en cada camino de propagación.
c. La polarización se inicializará en 1 en todas las neuronas, como valor conceptual adoptado para este parámetro.

12. Se Entrena la RNA con ejemplos resultantes de la clasificación estadística, donde la entrada es la información de la matriz de referencia y la salida esperada de la red es el código generado en la búsqueda en amplitud sobre el grafo.
13. Enviar la información de la matriz de referencia hacia la RNA previamente entrenada
14. Obtener el código por cada trama.
15. Concatenar los códigos.
16. Comparar el código obtenido con el código ASCII, calculo teórico y Huffman clásico.

3.4.3 Resultados de las Pruebas

En este apartado se exponen los resultados y análisis de los mismo, obtenidos de las pruebas aplicadas al algoritmo. Estos resultados están agrupados con respecto a las variables que inciden en el proceso; luego de esto se evalúan de forma consolidada y de esta forma se realizó un análisis conjunto.

3.4.3.1 Cálculo Teórico de la Tasa de compresión

Un referente importante para este tipo de investigaciones es la teoría de la información postulada por Claude E. Shannon; en ella explica la noción de información, en la que Shannon estableció resultados matemáticos acerca de los recursos que se necesitan para la codificación óptima y para la comunicación libre de errores.

La parte que hace referencia a la codificación explica que una fuente de información S es un sistema que contiene un conjunto de estados diferentes $(s_1, s_2, s_3, \dots\dots\dots, s_n)$, llamados

usualmente símbolos del alfabeto. Un aspecto central de la teoría de Shannon es que es posible asignar probabilidades de ocurrencia para los distintos estados de la fuente. Es decir, los estados s_n son producidos con probabilidades $P(s_1), P(s_2), P(s_3), \ldots\ldots\ldots P(s_n)$, entonces la cantidad de información generada por la fuente debido a la ocurrencia del estado s_i se define como:

$$I(S) = -[\log_2 P(s_i))] \quad (3.44)$$

Dado que S produce sucesiones de estados, estas sucesiones son usualmente llamadas mensajes, la entropía de la fuente S se define como la cantidad promedio de información producida por la fuente:

$$H(S) = -\sum_{i=1}^{n} P(s_i)\ \log_2 P(s_i) \quad (3.45)$$

En el contexto de la teoría de la información de Shannon, codificar implica establecer un mapeo entre los símbolos del alfabeto de la fuente S y el conjunto de cadenas de longitud finita de símbolos del alfabeto del código A. Estas suelen llamarse palabras-código. En general, las palabras-código no tienen la misma longitud. Cada palabra-código W que corresponde a la letra s_i, va a tener una longitud L_{s_i} , pero las longitudes de cada una de las palabras-código pueden variar. Entonces se define una longitud de palabra-código promedio como:

$$L = \sum_{i=1}^{n} P(s_i)\ l_i \quad (3.46)$$

L es entonces una medida de cuánto puede compactarse el código. En otras, palabras, un código con un valor de L más pequeño será más eficiente, dado que economiza más recursos en la transmisión. El Teorema del Canal Sin ruido afirma que existe un proceso de codificación óptimo tal que la longitud de palabra-código promedio L está tan cerca como se quiera del límite inferior $Lmin$ para L, esta definición se le conoce como el límite de la capacidad de canal de Shannon:

$$L_{min} = \frac{H(s)}{\log_2 2} = H(S) \quad (3.47)$$

Shannon plantea que existe un límite a la eficiencia en la codificación de la fuente. Si puede determinarse la entropía de una fuente caracterizada por la emisión de un número finito de símbolos, esto para codificadores con procedimientos estadísticos, entonces sabemos que H (en bits/símbolo) equivale al mínimo número de dígitos binarios que podrían emplearse para su codificación; en otras palabras H equivale al mínimo valor de la longitud media de cada símbolo de las palabras código.

H (S) se mide en bits y el logaritmo está en base 2 ya que en el lenguaje binario solo hay dos símbolos.

A continuación, se muestra la tabla 9, en la cual se relacionan los valores obtenidos para los cálculos teóricos del valor teórico de compresión basados en los cálculos de Shannon.

Tabla 9 Pruebas Calculo Teórico de Shannon

		Calculo Teórico de Shannon		
Prueba	**N° Pixeles**	**N° Colores**	**N° Bits $I(S)$**	**Entropía Media $H(S)$**
1	86400	11437	18684	5,73587963
2	86400	29800	98860	4,645775463
3	86400	10546	168034	10,18586806
4	86400	7309	115029	10,12746528
5	86400	14037	173422	12,10677083
6	86400	971	12275	8,673865741
7	86400	9004	141411	9,310023148
8	86400	11368	174075	12,50481481

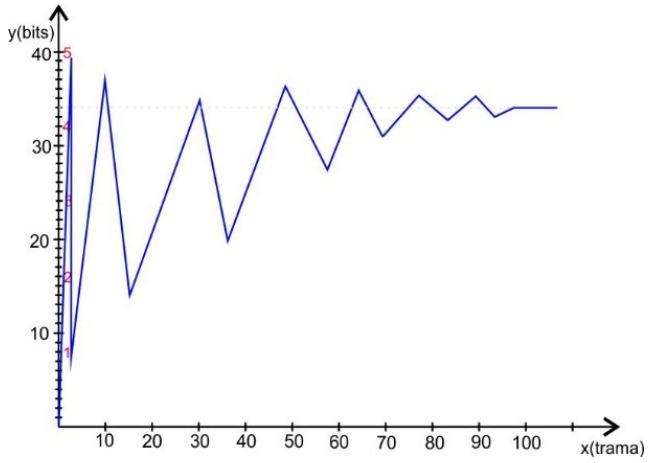
En la tabla anterior aparecen la cantidad de colores encontrados en cada video de prueba, así como la cantidad de bits promedio por trama y su respectiva entropía; como se analiza de los valores de la tabla, la longitud y por ende su entropía no solo depende de la cantidad de colores que aparecen, sino también de la probabilidad de ocurrencia de cada color.

3.4.3.2 Resultados y análisis por Video

Prueba 1

Tabla 10 Resultados de la Prueba 1

TIPO DE VIDEO				1
ITERACIÓNES				11
UNIDAD	**ASCII**	**TEORICO $I(S)$**	**HUFFMAN**	**HUFFMAN MODIFICADO**
Bits / Trama	167624	18684	41906	26615
Bytes / Trama	20953,00	2335,50	5238,25	3326,88
Diferencia con la tasa de Compresión Teórica $C_{\frac{H}{Sh}}$ (%)				42,45
Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)				0,635111917
Complejidad $T(n)$ (Unidades tiempo)				520026

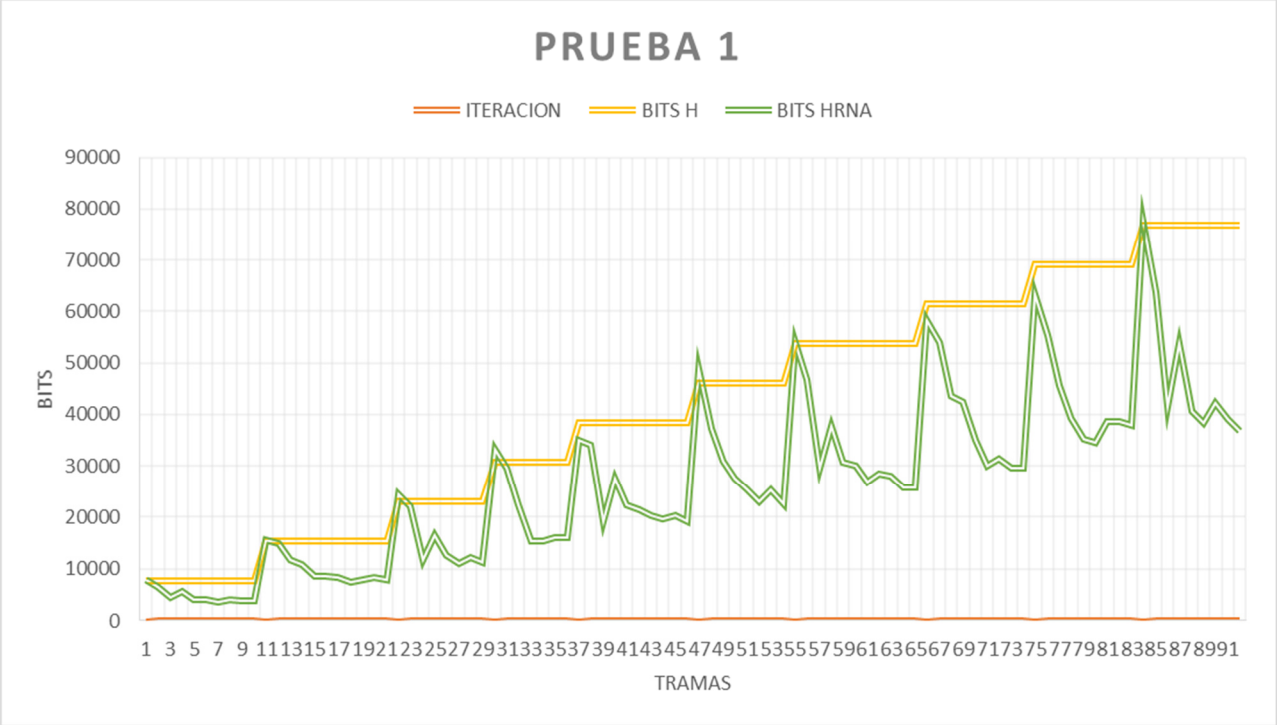


Gráfica 29 Prueba 1 Bits vs Trama

La Gráfica anterior (29) y las siguientes similares, describen como cambia la cantidad de bits en cada una de las tramas que salen del codificador; el eje horizontal representa las tramas de las que está compuesto el video y el eje vertical el número de bits que se generan en el codificador por la trama asociada.

Los fotogramas del video cambian frecuentemente, por esta razón el algoritmo no reconoce patrones dentro de las tramas de 5 bytes, de forma consecutiva; la variabilidad de las cadenas de bits hace que los símbolos se repitan con frecuencias similares, excepto por algunos que tienen muy baja probabilidad (códigos más largos), otros con probabilidades altas (códigos cortos), pero la mayoría de las tramas se sitúa en longitudes alrededor de 34 bits.

Con respecto a la codificación ASCII, la cadena de código Huffman es más corta con una relación de compresión de 0.85; lo que evidencia que se realiza una compresión sobre la información reduciendo en 15% su tamaño original.

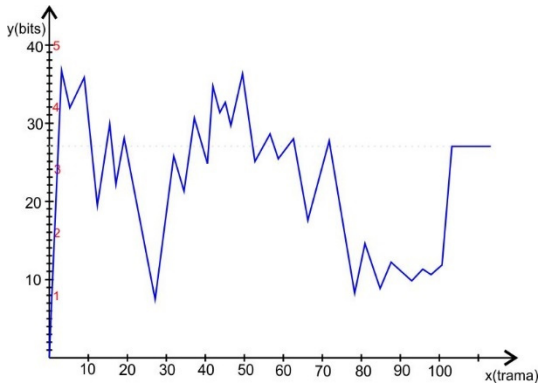


Gráfica 30 Prueba 1 Códigos Huffman vs Modificación Huffman

Prueba 2

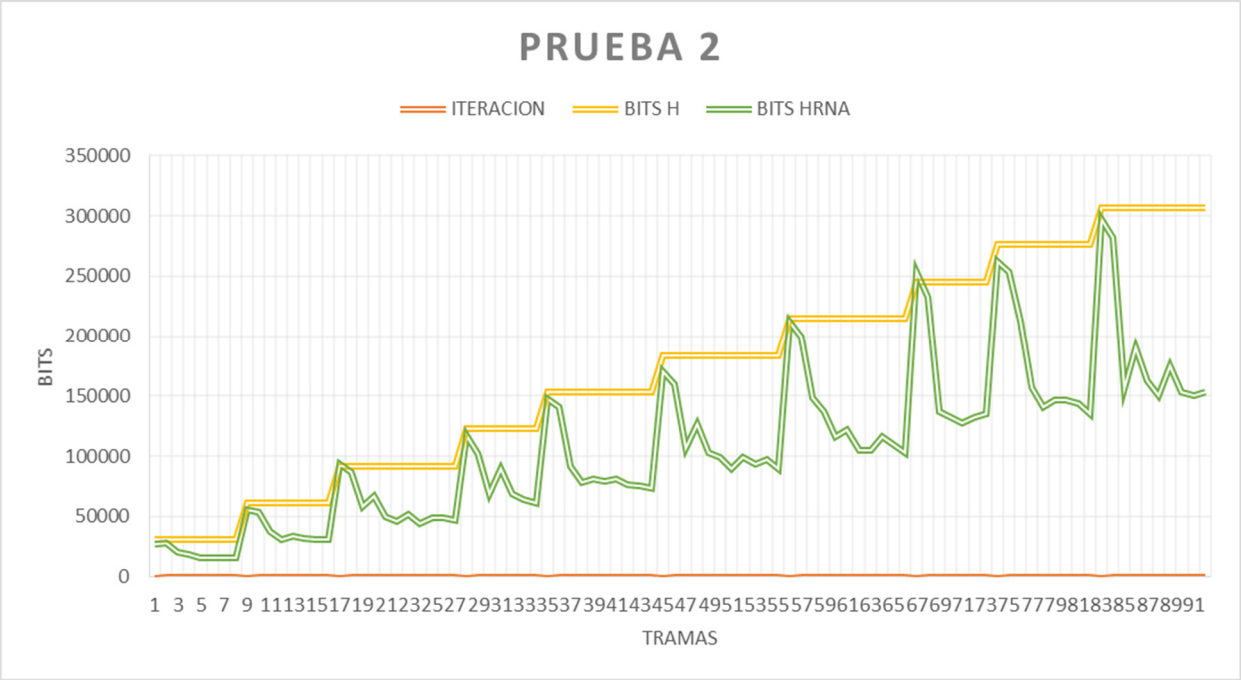
Tabla 11 Resultados de la Prueba 2

TIPO DE VIDEO				2
ITERACIÓNES				11
UNIDAD	**ASCII**	**TEORICO**	**HUFFMAN**	**HUFFMAN MODIFICADO**
Bits / Trama	511233	98860	171650	107996
Bytes / Trama	63904,13	12357,50	21456,25	13499,50
Diferencia con la tasa de Compresión Teórica $C_{\frac{H}{Sh}}$ (%)				9,24
Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)				0,629163997
Complejidad $T(n)$ *(Unidades tiempo)*				61852



Gráfica 31 Prueba 2 Bits vs Trama

En la prueba con el segundo video, se puede observar que el tamaño de las tramas varía mucho de una a la otra, esto se debe a que en la primera parte del video no se genera muchos cambios de un fotograma a otro, lo que conlleva a que tramas tengan probabilidades altas y longitudes cortas; cuando aparece un fotograma con muchos cambios la probabilidad cambia para los nuevos símbolos y así aumenta su longitud; el segmento de video que no tiene tantos cambios sesga el promedio de la longitud por trama y así mejora la taza de compresión con respecto al código ASCII.

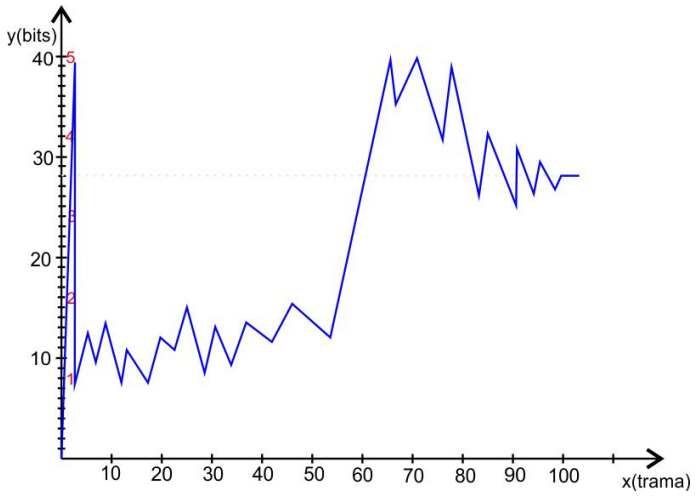


Gráfica 32 Prueba 2 Códigos Huffman vs Modificación Huffman

Prueba 3

Tabla 12 Resultados de la Prueba 3

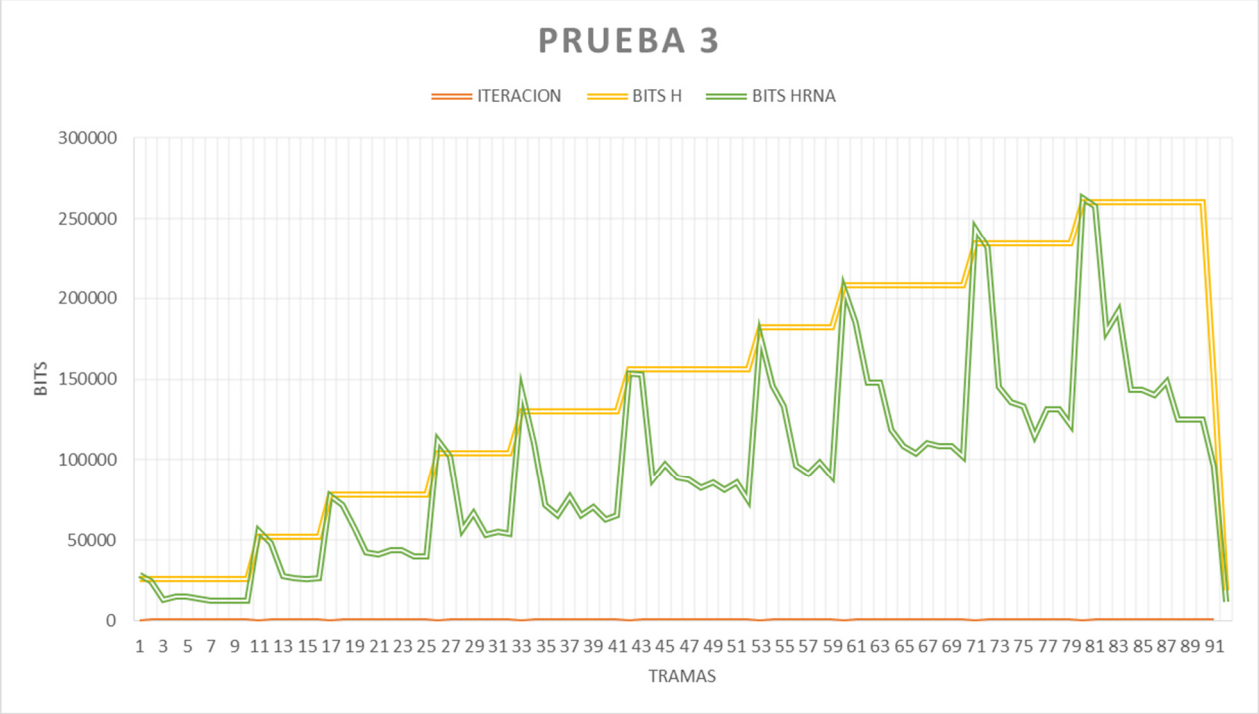
TIPO DE VIDEO				3
ITERACIÓNES				11
UNIDAD	**ASCII**	**TEORICO**	**HUFFMAN**	**HUFFMAN MODIFICADO**
Bits / Trama	461943	88034	149126	96005
Bytes / Trama	57742,88	11004,25	18640,75	12000,63
Diferencia con la tasa de Compresión Teórica $C_{\frac{H}{Sh}}$ (%)				9,05
Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)				0,643784451
Complejidad $T(n)$ *(Unidades tiempo)*				440154



Gráfica 33 Prueba 3 Bits vs Trama

En la primera parte del video los fotogramas no cambian mucho de uno a otro por esta razón, el número de símbolos no aumenta y sus probabilidades altas disminuyen la longitud de las cadenas de bits; en la parte final del video ocurre todo lo contrario la variación significativa de un fotograma a otro hace que las longitudes aumenten.

La tasa de compresión sigue siendo mejor que la de códigos ASCII, pero desmejoró con respecto al video anterior.

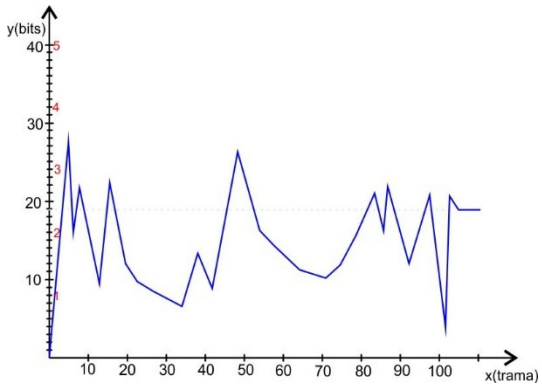


Gráfica 34 Prueba 3 Códigos Huffman vs Modificación Huffman

Prueba 4

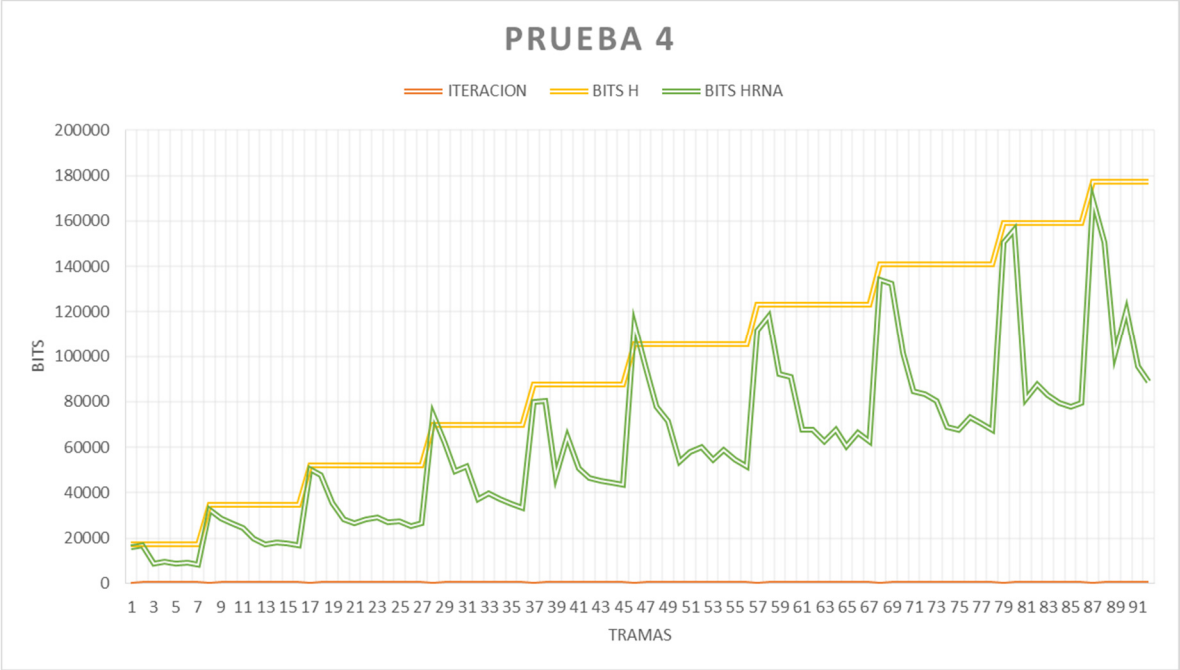
Tabla 13 Resultados de la Prueba 4

TIPO DE VIDEO				4
ITERACIÓNES				11
UNIDAD	**ASCII**	**TEORICO**	**HUFFMAN**	**HUFFMAN MODIFICADO**
Bits / Trama	273495	50034	98407	62388
Bytes / Trama	34186,88	6254,25	12300,88	7798,50
Diferencia con la tasa de Compresión Teórica $C_{\frac{H}{Sh}}$ (%)				24,69
Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)				0,63397929
Complejidad $T(n)$ *(Unidades tiempo)*				190359



Gráfica 35 Prueba 4 Bits vs Trama

Al ingresar un video homogéneo de un fotograma a otro, los símbolos tienen más altas probabilidades y menor número de tramas con valores de probabilidad distintos de cero; por tanto, las cadenas son más cortas y se repiten más que en las anteriores pruebas, lo que genera una tasa de compresión mejor con respecto al código de referencia.



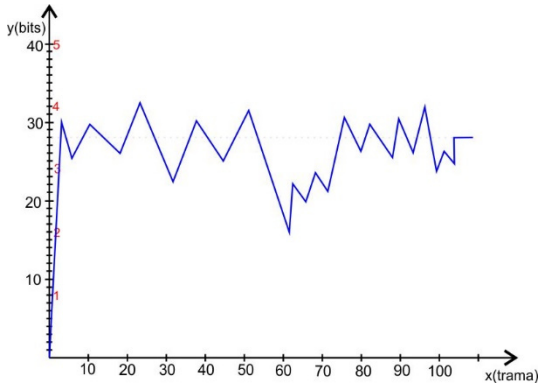
Gráfica 36 Prueba 4 Códigos Huffman vs Modificación Huffman

Prueba 5

Tabla 14 Resultados de la Prueba 5

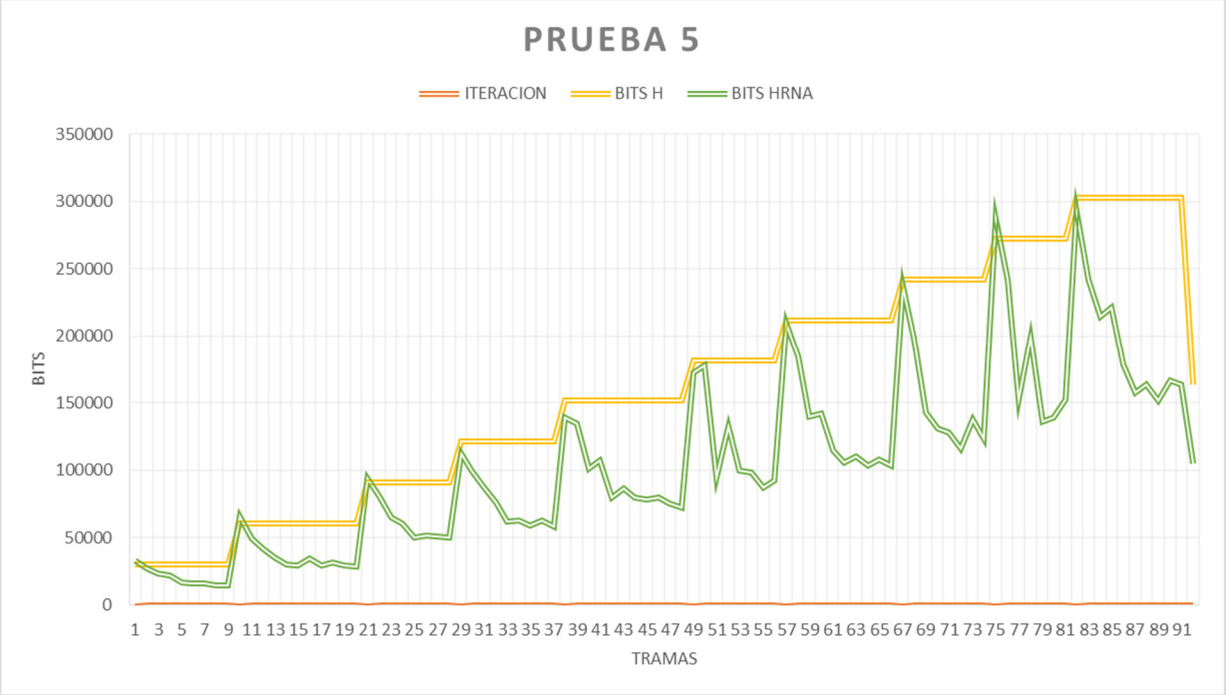
TIPO DE VIDEO	1
ITERACIÓNES	11

UNIDAD	ASCII	TEORICO	HUFFMAN	HUFFMAN MODIFICADO
Bits / Trama	496211	89265	163300	105113
Bytes / Trama	62026,38	11158,13	20412,50	13139,13
Diferencia con la tasa de Compresión Teórica $C_{\frac{H}{Sh}}$ (%)				17,75
Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)				0,643680343
Complejidad $T(n)$ *(Unidades tiempo)*				440226



Gráfica 37 Prueba 5 Bits vs Trama

Con la RNA ya entrenada durante las pruebas anteriores utilizando Backpropagation y al pasar el video (prueba 5) se nota significativamente la mejora, ya que el algoritmo está preparado para encontrar patrones en la secuencia de tramas cada vez más largas y así disminuir la longitud aumentando la probabilidad de aparición de algunos patrones presentes en los fotogramas.

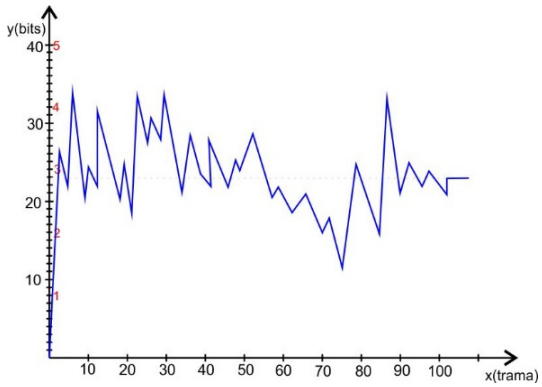


Gráfica 38 Prueba 5 Códigos Huffman vs Modificación Huffman

Prueba 6

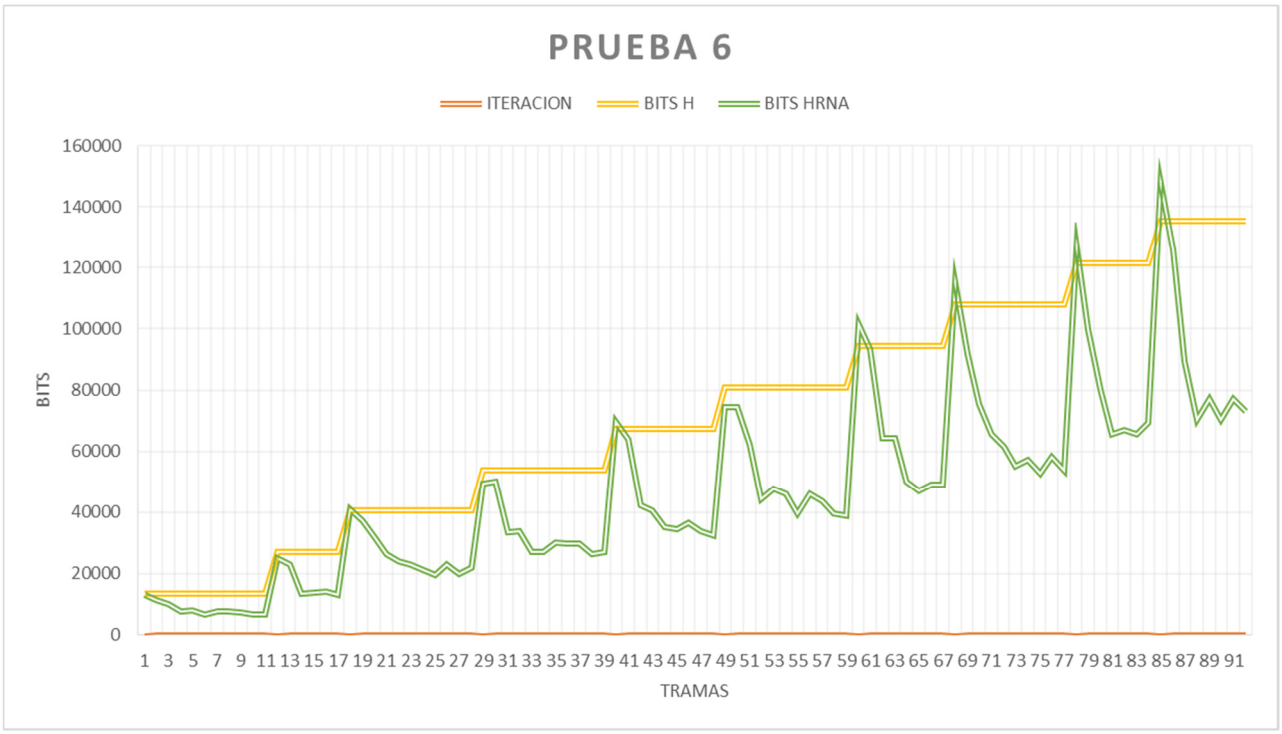
Tabla 15 Resultados de la Prueba 6

TIPO DE VIDEO				2
ITERACIÓNES				11
UNIDAD	**ASCII**	**TEORICO**	**HUFFMAN**	**HUFFMAN MODIFICADO**
Bits / Trama	218867	39265	73852	47474
Bytes / Trama	27358,38	4908,13	9231,50	5934,25
Diferencia con la tasa de Compresión Teórica $C_{\frac{H}{Sh}}$ (%)				20,91
Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)				0,642826193
Complejidad $T(n)$ *(Unidades tiempo)*				213018



Gráfica 39 Prueba 6 Bits vs Trama

Debido a la naturaleza del video en su material audiovisual, la tasa de compresión disminuye notablemente casi hasta ser la mitad de la codificación ASCII, con respecto a la codificación del mismo video en la primera iteración, se puede notar la mejora, al igual que lo ocurrido en el video tipo 1 y de la misma forma la RNA entrenada está en capacidad de mejorar el reconocimiento de los símbolos y su codificación agrupando y disminuyendo la longitud en bits de las tramas.

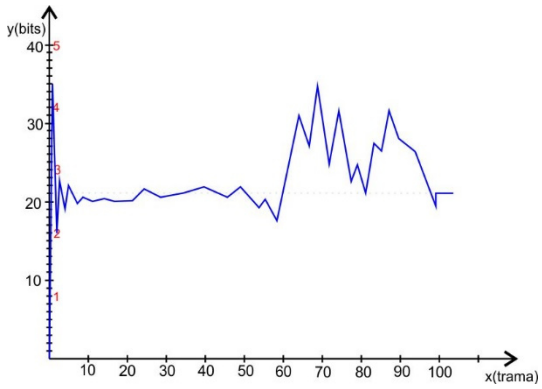


Gráfica 40 Prueba 6 Códigos Huffman vs Modificación Huffman

Prueba 7

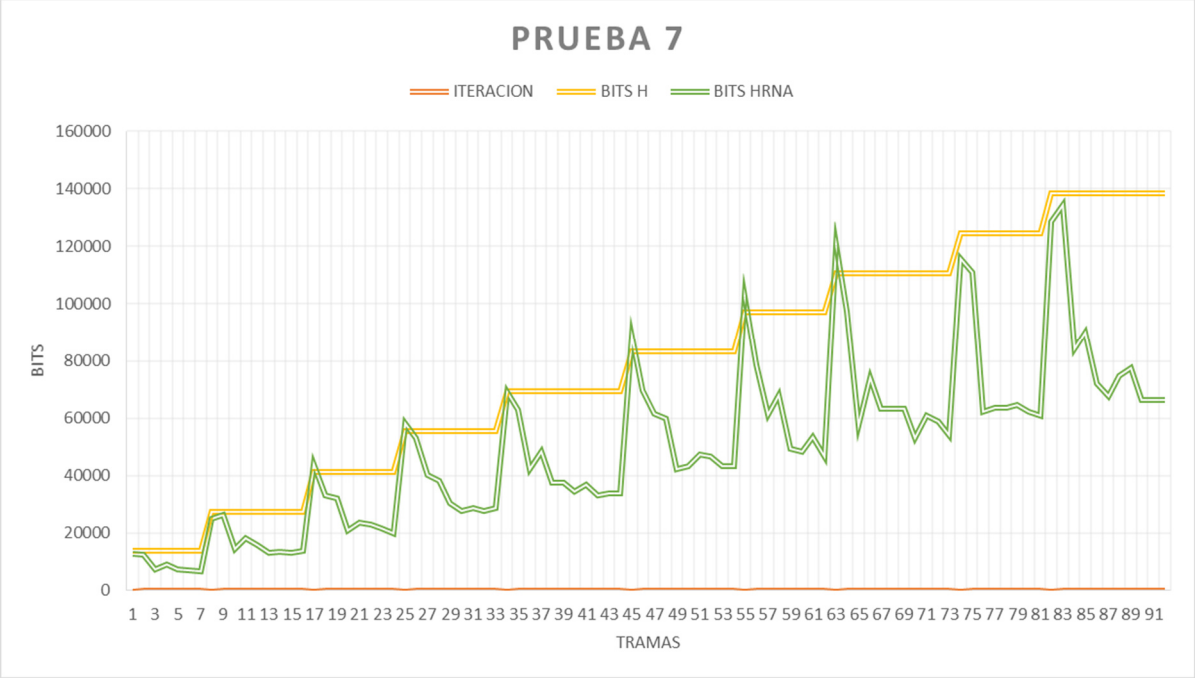
Tabla 16 Resultados de la Prueba 7

TIPO DE VIDEO				3
ITERACIÓNES				11
UNIDAD	**ASCII**	**TEORICO**	**HUFFMAN**	**HUFFMAN MODIFICADO**
Bits / Trama	246119	42265	79098	49902
Bytes / Trama	30764,88	5283,13	9887,25	6237,75
Diferencia con la tasa de Compresión Teórica $C_{\frac{H}{Sh}}$ (%)				18,07
Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)				0,630888265
Complejidad $T(n)$ *(Unidades tiempo)*				209703



Gráfica 41 Prueba 7 Bits vs Trama

Como el video de tipo dos, el video de tipo tres tiene una distribución que va de la simplicidad a la complejidad en sus fotogramas, por esta razón en la segunda iteración mejora aún más que en la iteración anterior y comparado con su análogo de tipo dos en la misma iteración reduce la tasa de compresión, aunque en un valor no muy amplio.

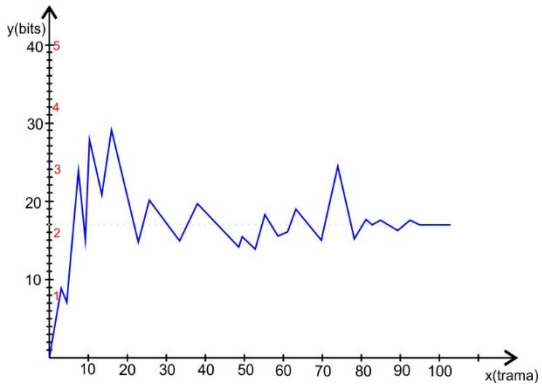


Gráfica 42 Prueba 7 Códigos Huffman vs Modificación Huffman

Prueba 8

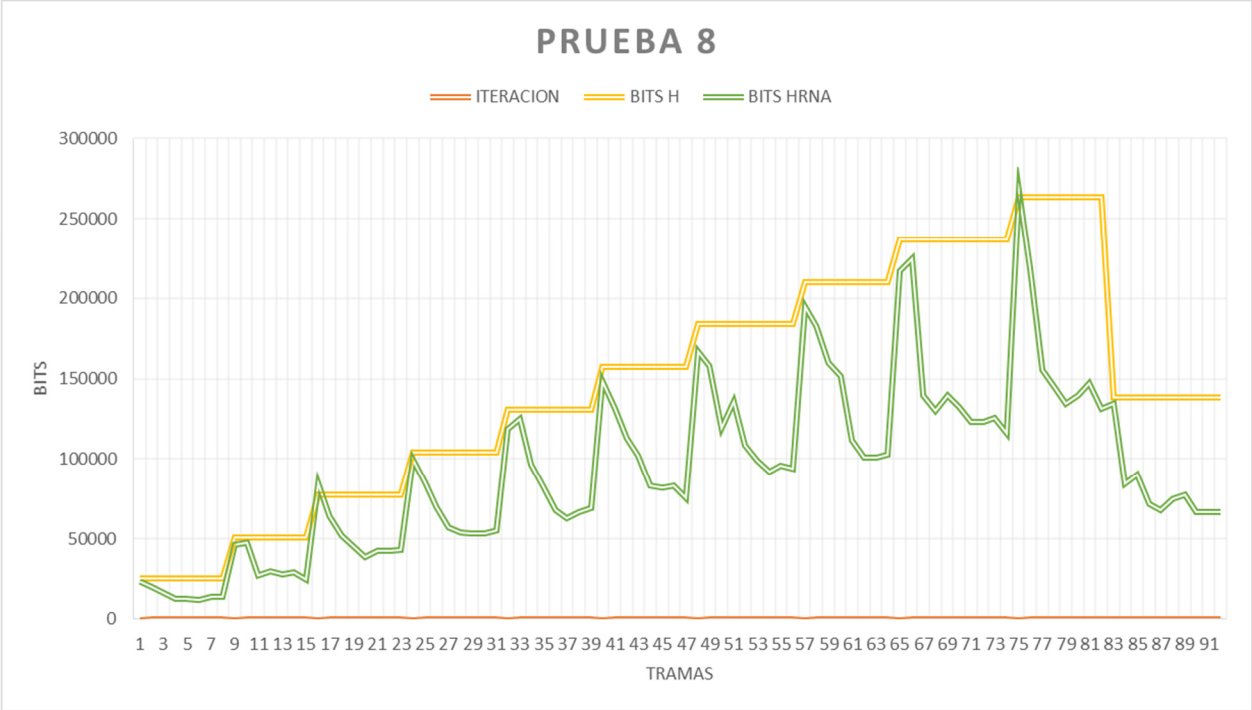
Tabla 17 Resultados de la Prueba 8

TIPO DE VIDEO				4
ITERACIÓNES				11
UNIDAD	**ASCII**	**TEORICO**	**HUFFMAN**	**HUFFMAN MODIFICADO**
Bits / Trama	428728	85265	146760	93351
Bytes / Trama	53591,00	10658,13	18345,00	11668,88
Diferencia con la tasa de Compresión Teórica $C_{\frac{H}{Sh}}$ (%)				9,48
Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)				0,636079313
Complejidad $T(n)$ *(Unidades tiempo)*				166023



Gráfica 43 Prueba 8 Bits vs Trama

El resultado de compresión en la codificación para este video es el esperado, ya que por las características propias y la mejora en el entrenamiento en la segunda iteración garantizan una tasa de compresión mejorada con respecto a la iteración anterior y a los demás videos que se prueban.



Gráfica 44 Prueba 8 Códigos Huffman vs Modificación Huffman

3.4.3.3 Resultados y análisis por Iteración

Cada una de las pruebas realizadas, sobre cada uno de los tipos de video, se pasa sobre la RNA en varias ocasiones (Iteraciones), con el objetivo de que la red misma se entrene y reduzca el error medio en el trazo del grafo generado por el algoritmo; de esta manera, los resultados de la codificación de una misma trama de información pueden variar con respecto al número de iteraciones.

Lo que se busca al realizar estas pruebas sobre las tramas correspondientes es determinar la cantidad de iteraciones necesarias para obtener una mejor codificación. De allí se obtiene el valor de las mejores tasas de compresión que puede ofrecer la red neuronal y qué cantidad de iteraciones son necesarias para obtenerlas; de esta manera se puede calcular la complejidad temporal, aproximándose a la cantidad de recursos computacionales necesarios.

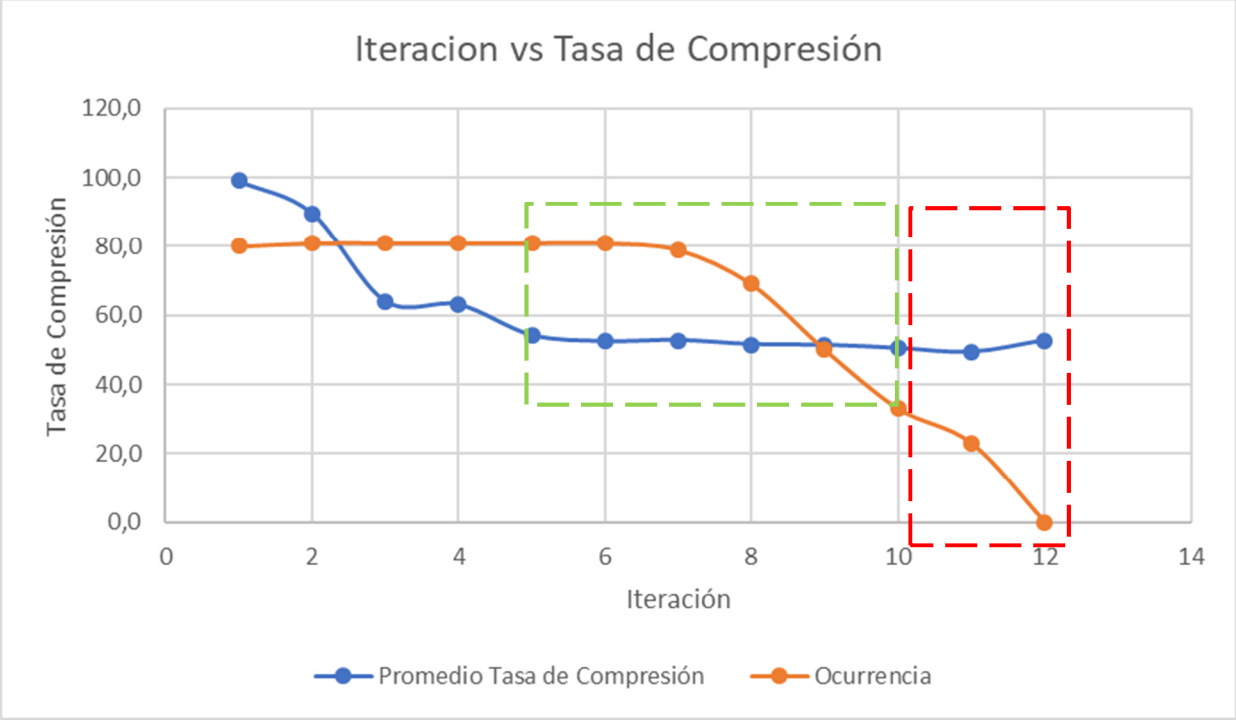
Tabla 18 Resultados de las pruebas por Iteración

Iteración	Promedio Tasa de Compresión $C_{\frac{H}{A}}$ (%/100)	Ocurrencia
1	0,990	0
2	0,897	0
3	0,640	0
4	0,630	0
5	0,542	81
6	0,524	81
7	0,527	79
8	0,515	69
9	0,513	50
10	0,504	33
11	0,493	23
12	0,527	0

En la tabla anterior (18), se relaciona la tasa de compresión promedio que brinda la RNA en cada una de las iteraciones. En la última columna de la tabla se muestra la ocurrencia; que indica el número de veces que se encontró el mejor código en cada iteración. Como se observa de las iteraciones de la 1 a la 4, la ocurrencia es cero, de esto se deduce que los códigos encontrados en estas iteraciones se pueden mejorar, también se observa que la

cantidad mínima de iteraciones que deben realizarse para encontrar la mejor codificación es 5; igualmente qué la ocurrencia va disminuyendo, con lo que se puede concluir que después de la quinta iteración es menos probable que se encuentre una mejor codificación, por esta razón se señala el intervalo entre la iteración 5 y la iteración 10, rango en el cual se encuentran las mejores codificaciones, representadas en las tasas de compresión, que varían entre el 54,2% y el 50,4, con una diferencia del 3.8 %, con una media de mejora por iteración de 0.76%. Lo anterior para señalar que de la iteración 1 a la 5 es mucho más significativa la mejora. que pasa de 99,0% en la iteración 1 a 54,2% en la iteración 5, con una media de mejora por iteración de 8.96%.

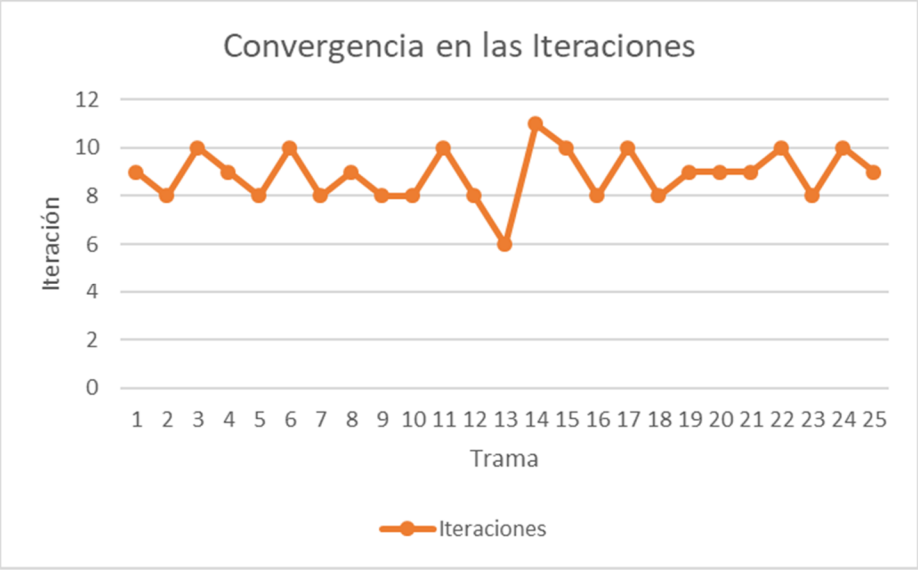
Lo anterior demuestra que en las primeras 5 iteraciones la red neuronal mejora significativamente la tasa de compresión y por ende la codificación; mientras que de la iteración 5 hacia adelante el valor de la tasa de compresión tiende a converger en un valor promedio de 64%, valor que se puede extraer de las pruebas realizadas por cada video, descrita en el apartado anterior.



Gráfica 45 Iteraciones vs Tasas de Compresión

En la imagen (grafica 45), se aprecia lo descrito en el párrafo anterior, en la cual se representan las funciones de número de iteraciones versus tasas de compresión. Como se describió, la tasa de compresión promedio por iteración, mejora significativamente en el intervalo de las iteraciones 1 a 5 y luego tiende a variar poco mientras se ejecutan de 5 a 10 iteraciones. Si se observa la tabla 18, se puede ver que la iteración número 11, presenta una tasa de compresión mejor que la iteración 10, la razón por la que no se incluye en el intervalo que se estudia, es que la ocurrencia tiene un valor de 23, lo que determina que de las 739 tramas en total solo en 23 ocasiones se encontró un mejor código en esta iteración; por lo que es muy poco probable que en las iteraciones mayores a 10 se encuentre una codificación mejor que las encontradas en el intervalo de la quinta a la décima iteración, pero llegar a una onceaba iteración si representa un costo alto de complejidad temporal en comparación con la diferencia de la tasa de compresión que se puede obtener, que es en promedio de 1.1%.

La misma conclusión se puede observar en la gráfica 45, donde se aprecia que la tasa de compresión se maneja de una forma casi constante debido a que no sufre un cambio muy significativo entre las iteraciones 5 a 10, caso contrario en la ocurrencia por iteración de la gráfica 45, donde se ve claramente que es una gráfica descendente en los valores superiores a 7.



Gráfica 46 Comportamiento Convergente de las Iteraciones

Lo anterior se puede resumir en la gráfica 46, en la que se muestra la función de iteración versus la trama; en esta función se colocó el número de iteración en la cual se encontró el mejor código de cada una de las tramas; en esta se aprecia que la búsqueda de la mejor codificación se ubica alrededor de la iteración 8, con un mejor desempeño en la iteración 6 y el peor en la iteración 11.

3.4.3.4 Resultados y análisis por tasa de compresión

Para realizar las pruebas en el algoritmo, los archivos de video se dividieron en tramas, cada una con duración de tres segundos; estas tramas se procesaban por el codificador y se obtenían los códigos resultantes, medidos en su longitud o cantidad de bits que los conforman; se utilizó codificación ASCII y Huffman, como referencia para la comparación con el código obtenido del Huffman Modificado. La tasa o razón de compresión se obtiene con la siguiente ecuación:

$$T_C = \frac{H_{RNA}}{H_s} 100 \qquad (3.48)$$

Lo que representa es la cantidad porcentual del tamaño del código de Huffman modificado con respecto al código Huffman canónico.

La información de la segmentación del video se muestra en la siguiente tabla:

Tabla 19 Información de segmentación de los videos de prueba

Concepto	Valor	Unidad
Duración	30	Segundos
Resolución	1280x720	Píxel x Píxel
Numero de pixeles por Fotograma	921600	Píxel
Velocidad fotograma	24	Frames/segundo
Tramas por Video	10	Trama/Video
Duración de la Trama	3	Segundos
Fotogramas por Trama	72	Frames/Trama
Pixeles por Trama	66355200	Pixel/Trama

Cada una de las tramas se procesa con el algoritmo de Huffman canónico y en paralelo con el algoritmo de Huffman modificado con la RNA, obteniéndose los códigos respectivos, con la diferencia que en el segundo método se itera sobre la red neuronal un número dinámico

de veces hasta que el algoritmo encuentra la mejor codificación. Esto último documentado en el apartado anterior. Cada código obtenido en cada trama se registró con el fin de construir una tabla de datos agrupados para determinar las mejores y más probables tasas de compresión obtenidas al finalizar las pruebas.

Para realizar la construcción de la tabla de datos agrupados se obtuvieron los siguientes valores de referencia:

Tabla 20 Valores Estadísticos de referencia

Variable	Valor
Datos de Entrada: tramas (Muestra)	739
Máximo Valor de la Tasa de Compresión (%)	110
Mínimo Valor de la Tasa de Compresión (%)	48
Media Aritmética (%)	63.8
Desviación Estándar (%)	17,66482077
Rango = Max – Min (%)	62
Numero de Intervalos $= 1 + 3.32(\log Datos)$	11
Longitud del Intervalo = Rango / Numero de Intervalos	6

Un dato importante es la longitud del intervalo, ya que define el intervalo de compresión sobre el cual se realizó el análisis. A continuación, se muestra la tabla obtenida.

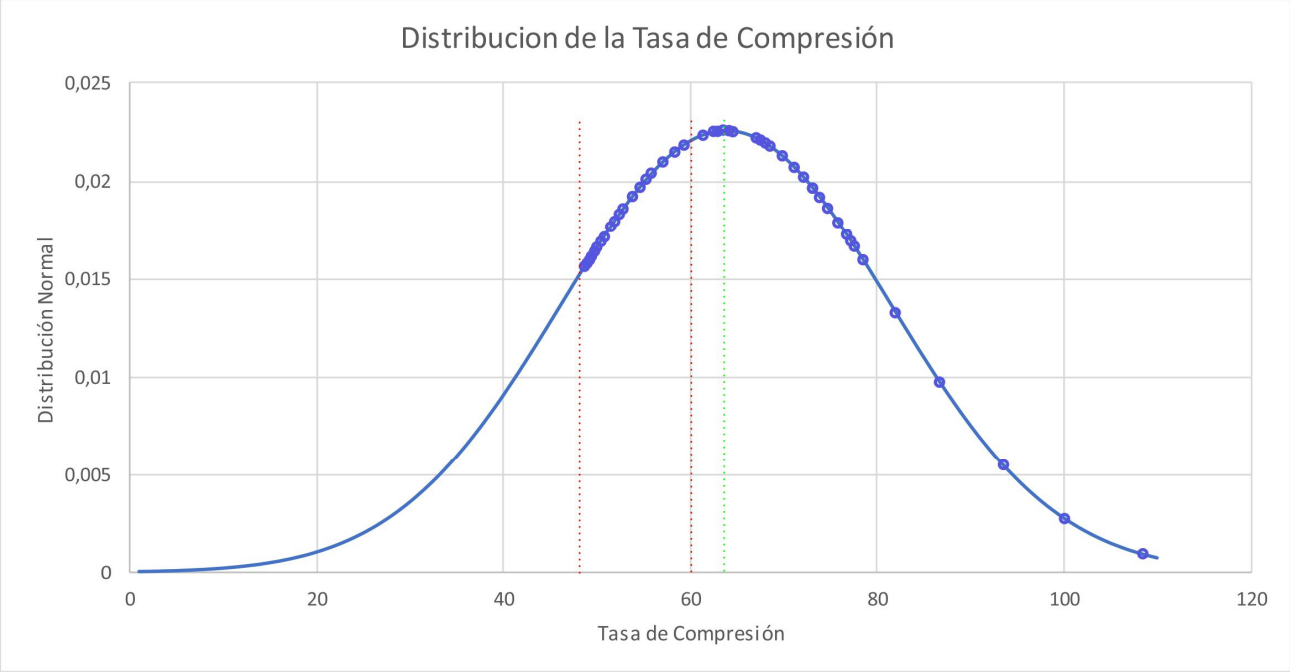
Tabla 21 Resultados de las Pruebas en Datos Agrupados

N° Del Grupo	Grupo/ Clase		Marca Grupo (Valor medio)	Frecuencia Absoluta	Frecuencia Relativa	Frecuencia Porcentual (%)
	Lim Inferior	Lim Superior				
0	0	48	24,0	0	0,000	0,00
1	48	54	50,9	343	0,464	46,41
2	54	60	56,8	134	0,181	18,13
3	60	66	62,7	30	0,041	4,06
4	66	72	68,6	42	0,057	5,68
5	72	77	74,5	27	0,037	3,65
6	77	83	80,4	18	0,024	2,44
7	83	89	86,3	23	0,031	3,11
8	89	95	92,2	54	0,073	7,31
9	95	101	98,1	37	0,050	5,01
10	101	107	103,9	19	0,026	2,57
11	107	110	108,4	12	0,016	1,62
				739	1	100

De los resultados representados en la tabla anterior (21), se pueden analizar las siguientes conclusiones:

- Las mejores tasas de compresión clasificadas por su probabilidad de ocurrencia se encuentran en el primer y segundo grupo en el rango de 48% a 60%; aunque en el intervalo de 48% a 54% es realmente el valor significativo. De esto se puede interpretar que es muy probable encontrar códigos con tasa de compresión de alrededor de 50.9%.
- En los grupos que tienen tasas de compresión entre los 60% y 110%, tienen muy poca probabilidad de ocurrencia; con lo que se interpreta que el algoritmo si converge a un mejor código cada vez que se utiliza y es más rápido si la RNA está más entrenada.
- Si se cruza la información de la tabla 18 y la tabla 21, se observa que los valores de tasa de compresión, que se encuentran en el rango de 40% a 54%, son los valores que se obtienen si la RNA se ha ejecutado un numero de iteraciones entre 5 y 10, como ya se había identificado en el apartado anterior. Lo que confirma que es más probable encontrar el mejor código entre las iteraciones 5 y 10 con un máximo de compresión de 48% y un mínimo de 54%, con lo que se establece un número mínimo de iteraciones en la codificación para encontrar el mejor código y contrastar el número máximo de iteraciones con respecto al consumo de recursos físicos necesarios (Complejidad Computacional).

Por último, se puede representar los valores obtenidos en una gráfica de distribución normal, para identificar gráficamente el intervalo donde se agrupan los códigos según su probabilidad.



Gráfica 47 Distribución de probabilidad de las tasas de compresión

3.4.3.5 Resultados y análisis de complejidad computacional

En el numeral 3.2.2.3, se calculó la complejidad computacional total que utiliza el algoritmo de codificación en tiempo de ejecución para realizar el procesamiento de un segmento de información, esta complejidad viene representada por la expresión

$$T(n) = 13n^2 + 26 \quad (3.49)$$

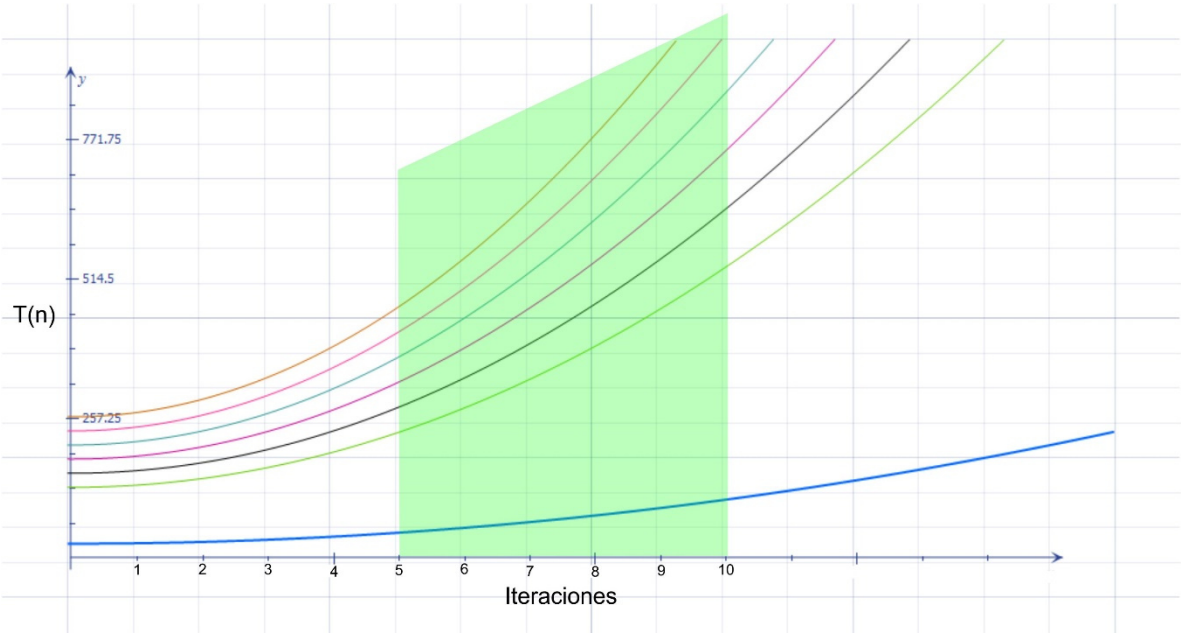
Es de aclarar que esta complejidad corresponde a una sola iteración de la RNA, por tal motivo la complejidad de una trama de información será un múltiplo de esta función que depende de la cantidad de iteraciones a realizar.

El valor de n depende de la cantidad de datos que ingresan al algoritmo, lo que lo convierte en un intervalo de valores de entre 1 y 256 valores diferentes que dependen de la cantidad de información de los fotogramas de una trama; con esta información se pueden calcular las cotas de complejidad de las funciones de complejidad según la cantidad de iteraciones a realizar en el intervalo de 5 a 10 iteraciones.

Tabla 22 Función de complejidad por iteración

Iteración	Complejidad
1	$T(n) = 13n^2 + 26$
5	$T(n) = 65n^2 + 130$
6	$T(n) = 78n^2 + 156$
7	$T(n) = 91n^2 + 182$
8	$T(n) = 104n^2 + 208$
9	$T(n) = 117n^2 + 234$
10	$T(n) = 130n^2 + 260$

Definidas las funciones de complejidad para cada una de las iteraciones, pueden determinarse las cotas en las cuales es probable que el algoritmo encuentre el mejor código y así contrastarlo con la cantidad de recursos necesarios para su ejecución.



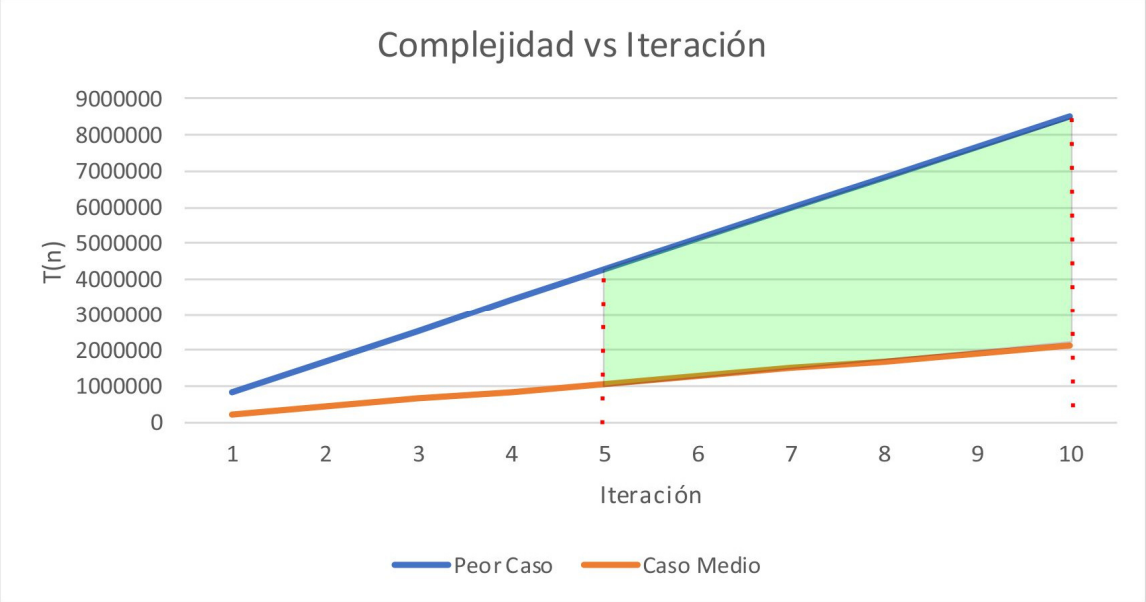
Gráfica 48 Complejidad Computacional vs Iteraciones

El área en verde de la gráfica anterior (48) define los limites en los cuales se puede obtener el mejor código y resalta la complejidad computacional del algoritmo en el rango de iteraciones con un aumento según la cantidad de información que se envía al codificador; es de anotar que todas las funciones de complejidad para el algoritmo son de orden cuadrático. Por esta razón es necesario tener los valores de la complejidad, para el caso de estudio se hicieron los cálculos del peor de los casos y caso medio de cada una de las iteraciones.

Tabla 23 Resultados de los cálculos de complejidad por iteración

	Complejidad		
Iteración	**Función**	**Peor Caso**	**Caso Medio**
1	$T(n) = 13n^2 + 26$	851994	213018
2		1703988	426036
3		2555982	639054
4		3407976	852072
5	$T(n) = 65n^2 + 130$	4259970	1065090
6	$T(n) = 78n^2 + 156$	5111964	1278108
7	$T(n) = 91n^2 + 182$	5963958	1491126
8	$T(n) = 104n^2 + 208$	6815952	1704144
9	$T(n) = 117n^2 + 234$	7667946	1917162
10	$T(n) = 130n^2 + 260$	8519940	2130180

De la información en esta tabla lo importante de resaltar, es la complejidad del rango comprendido entre las iteraciones 5 a la 10, para hacer una comparación entre la cantidad de una iteración a la siguiente.



Gráfica 49 Área útil de la linealización de la Complejidad vs Iteraciones

Como se observa en esta gráfica (49) el comportamiento de la complejidad es lineal, mientras que las cotas son cuadráticas; de lo anterior puede analizarse que, la complejidad puede predecirse para un determinado número de iteraciones y que esta predicción es independiente de la cantidad de información a la entrada del algoritmo.

Tabla 24 Comparación de complejidad entre las iteraciones una a una

Iteración	Diferencia

Inicio	Final	Peor Caso	Caso Medio
1	2	851994	213018
2	3	851994	213018
3	4	851994	213018
4	5	851994	213018
5	6	851994	213018
6	7	851994	213018
7	8	851994	213018
8	9	851994	213018
9	10	851994	213018

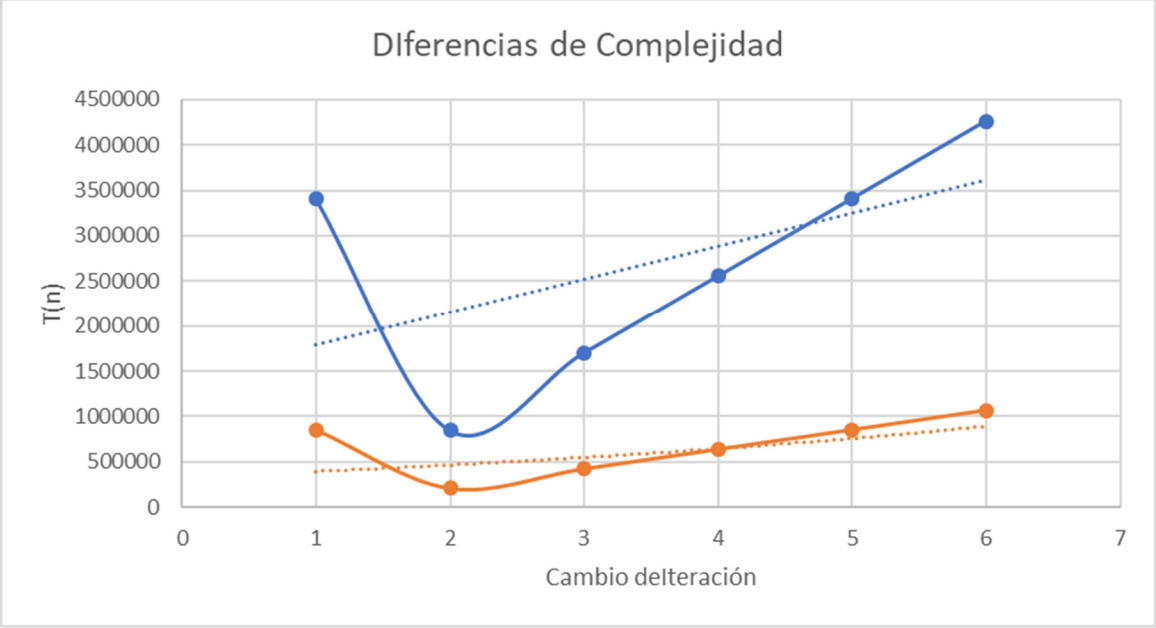
La información de esta tabla es muy interesante, debido a que, se puede interpretar que la complejidad y el cambio de iteración a iteración es constante, como era de esperarse debido a que la complejidad se comporta linealmente, con lo que se puede concluir que el aumento y la disminución de la complejidad solo depende de la cantidad de iteraciones que realice el algoritmo y no de la cantidad de elementos del alfabeto de entrada.

Otra forma de analizarla esto, es comparando iteración a iteración con respecto a la mejor iteración, en cuanto a complejidad se refiere, que para este caso es la iteración 5, hasta la peor iteración que sería la 10.

Tabla 25 Comparación de complejidad entre la mejor iteración hasta la ultima

Iteración		Diferencia	
Inicio	Final	Peor Caso	Caso Medio
1	5	3407976	852072
5	6	851994	213018
5	7	1703988	426036
5	8	2555982	639054
5	9	3407976	852072
5	10	4259970	1065090

Con los valores de las diferencias entre la iteración 5 y las demás iteraciones, se incrementan es necesario ver el comportamiento de la complejidad en una gráfica para determinar si el costo de la complejidad incide de manera negativa en la obtención de la mejor tasa de compresión.



Gráfica 50 Diferencias de Complejidad de Iteración a Iteración

En la gráfica 50 se observa que la diferencia entre la mejor iteración y las demás va aumentando de forma lineal con la misma pendiente que en la gráfica 49, por lo tanto se puede usar las pendientes de las rectas del peor caso y del caso medio como tendencia de comportamiento de la complejidad, corroborando que este valor solo depende de la iteración a la que se llegue a ejecutar el algoritmo, aclarando que esta complejidad es acumulativa y por esto el análisis se realizó con la diferencia de complejidades de una iteración a otra.

3.4.3.6 Análisis Integrado de los resultados

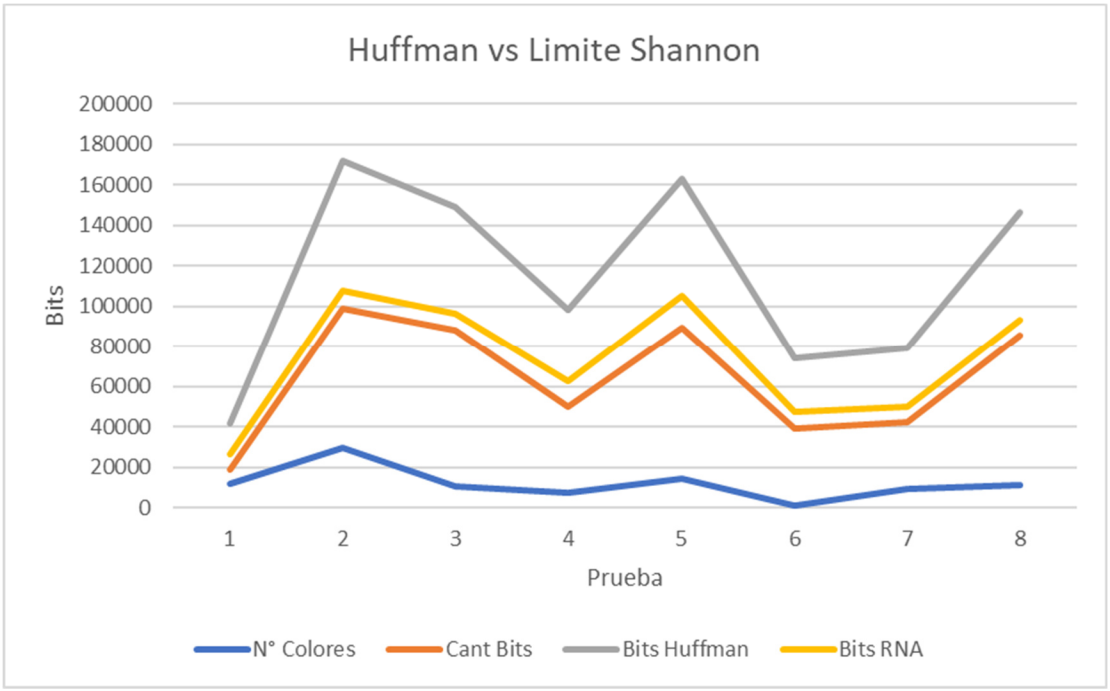
Los resultados del proceso de codificación y compresión de la información tienen un referente en la teoría de información de Shannon; por esta razón se realizó un análisis comparativo de cálculo teórico de la tasa de compresión con los postulados de Shannon con respecto a la tasa de compresión obtenida; los resultados se muestran en la siguiente tabla.

Tabla 26 Resultados de las pruebas en bits

	Calculo Teórico de Shannon			Huffman			
Prueba	N° Colores	Cant Bits	Entropía Media	Bits Huffman	Bits RNA	Tasa BRNA	Diferencia

1	11437	18684	5,73587963	41906	26615	142	42
2	29800	98860	4,645775463	171650	107996	109	9
3	10546	88034	10,18586806	149126	96005	109	9
4	7309	50034	10,12746528	98407	62388	125	25
5	14037	89265	12,10677083	163300	105113	118	18
6	971	39265	8,673865741	73852	47474	121	21
7	9004	42265	9,310023148	79098	49902	118	18
8	11368	85265	12,50481481	146760	93351	109	9

Del anterior análisis, se observa que la forma en la que se comportan el algoritmo clásico de Huffman y la modificación propuesta, es principalmente estadística y por tanto la teoría de Shannon, aplica completamente en ambos, demostrando que las tasas de compresión de ambos se comportan de manera similar, es de anotar que para este análisis se tomó las longitudes más cortas obtenidas de las pruebas realizadas sin discriminar por iteración.



Gráfica 51 Comparación entre los códigos Huffman y el límite de Shannon

En la gráfica anterior (51), se observa las formas similares en que se comportan los métodos de codificación, esto se debe a que todos se basan en los procesos de clasificación estadística y la probabilidad de ocurrencia de los símbolos que aparecen desde la fuente,

también se aclara que el número de bits de cada una de las pruebas depende en parte de la cantidad de colores que aparecen en el video pero en mayor medida de la probabilidad de cada color; entre más semejantes sean las probabilidades menos eficiente será la distribución de probabilidad.

Hasta el momento se ha realizado el análisis de los resultados en cuanto a las variables objeto de estudio en el desempeño del algoritmo propuesto, estas han sido con base en el tipo de información presentada al codificador, a la cantidad de veces que se ejecuta el módulo encargado de codificar mediante la RNA (iteración), el resultado de compactación del código o tasa de compresión y por último en el desempeño del algoritmo en tiempo de ejecución con respecto a los recursos necesarios en procesamiento, lo que se ha llamado en este trabajo, complejidad temporal o computacional; realizar un análisis conjunto de estas variables es lo que permitirá evaluar si esta propuesta mejora el proceso de codificación y compresión de la información y el costo que requiere para hacerlo.

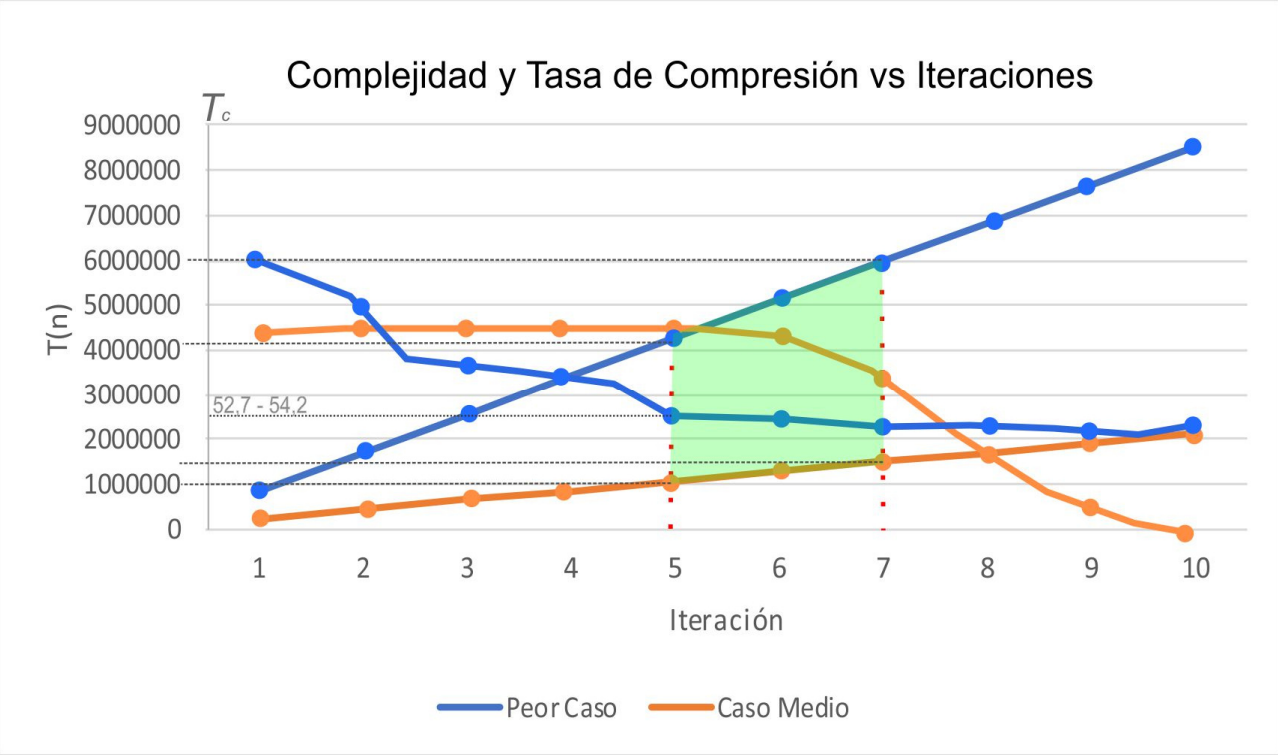
Como ya se vio anteriormente, la información no es un factor que se determinante en el desempeño del algoritmo, pese a su variación a la entrada del codificador, esta característica se da porque si la red ya fue entrenada o sigue entrenándose en cada iteración, la tasa de compresión mejora a pesar de que la complejidad aumente; por lo que la conjunción de estas tres variables, son la base de la evaluación de la propuesta.

Tabla 27 Resultados de las pruebas por complejidad

Iteración	Complejidad		Promedio Tasa de Compresión	Ocurrencia
	Peor Caso	Caso Medio		
1	851994	213018	99,0	80
2	1703988	426036	89,7	81
3	2555982	639054	64,0	81
4	3407976	852072	63,0	81
5	4259970	1065090	54,2	81
6	5111964	1278108	52,4	81
7	5963958	1491126	52,7	79
8	6815952	1704144	51,5	69
9	7667946	1917162	51,3	50
10	8519940	2130180	50,4	33

En la tabla se resaltan el rango de iteraciones entre la 5 y la 10, como se vio anteriormente en este rango se encuentran las mejores tasas de compresión con respecto a la probabilidad de ocurrencia de que en una iteración de este rango se encuentre el mejor código posible.

También se incluyen los valores de complejidad que causaría el uso de recursos en este rango de iteraciones, para determinar cuál es el mejor caso, teniendo en cuenta todas las variables tomadas en consideración.



Gráfica 52 Área útil de trabajo respecto a las tres variables

Como se observa en la gráfica 52, se cruzaron las tres variables en consideración, para establecer un área dentro del plano que representara las mejores condiciones para el codificador Huffman modificado.

De esta manera el grupo de mejores tasas de compresión se encuentra en el rango de las iteraciones de la 5 a la 10, todo esto con el fin de encontrar el código más probable dentro

de los mejores, a pesar de esto la mejor taza en este rango solo varia en un 1.5% valor poco representativo en el código que se obtiene.

Ubicado el rango de iteraciones con respecto a la tasa de compresión, lo siguiente es evaluar con respecto a la complejidad computacional; Como los dos casos a tener en cuenta son funciones lineales crecientes, es fácil observar que a medida que aumenta el número de iteraciones la complejidad también; sin embargo tomando en cuenta que de la iteración 7 en adelante la probabilidad de encontrar un mejor código disminuye rápidamente, se acota la gráfica en esta iteración, ya que no es necesario un gasto computacional más alto si no se tiene certeza de encontrar una mejor tasa de compresión.

Por todo lo anterior, en la gráfica 52 se muestra el área en color verde donde las tres variables son compensadas entre sí, de esta manera la probabilidad de encontrar un código con una tasa de compresión que oscila entre el 54% y 52%, dentro del rango de iteraciones entre la 5 y la 7 es de 0.645, con un costo computacional no elevado comparado con la tasa a obtener.

4. APORTES Y TRABAJOS FUTUROS

En el planteamiento de este trabajo de investigación, se consideró tener como aporte principal, la construcción de un codificador de la fuente de información, que usara como funcionamiento central el algoritmo de Huffman, realizar una modificación estructural a este procedimiento y así lograr comprimir la información a una mejor tasa que otros similares; sin embargo, en el desarrollo de la investigación y al hacer una inmersión profunda sobre el marco teórico conceptual en el que se basa este documento, surgieron algunos aportes que es importante mencionar.

El primero es haber desarrollado una metodología basada en ingeniería de software, para la implementación de un codificador de información, con el desarrollo de modelos estructurales y de comportamiento de alto nivel de abstracción, que permitan el desarrollo en cualquier plataforma o lenguaje de programación; de la misma forma mediante este

modelamiento, describir el valor de los recursos de procesamiento que se requieren y determinar la dependencia de este indicador de la cantidad de información y su diversidad.

A nivel estructural, la descripción de desglose mediante componentes del algoritmo, facilitó la interoperabilidad entre dos herramientas computacionales como lo son los modelos de programación y la computación evolutiva utilizando redes neuronales artificiales.

Otro aporte es la descripción matemática detallada de la RNA con entrenamiento Backpropagation, con el fin de lograr tener un control total sobre cada uno de sus componentes y así lograr que la red se comporte como una estructura de datos abstracta de forma dinámica a pesar de ser estática en su estructura, útil para remplazar el árbol binario del algoritmo clásico de Huffman y representar un grafo dirigido que permitiese encontrar mejores caminos para los códigos de Huffman encontrados.

También, encontrar la correlación no explícita entre las cuatro variables involucradas en la evaluación de los algoritmos utilizados para codificación y decodificación de información; de esta manera tener un indicador de desempeño con respecto a la relación costo beneficio entre recursos de procesamiento y tasa de compresión contra la cantidad de información en la entrada del codificador y el número de veces que se debe ejecutar la codificación para encontrar un código mejor.

Por último, es necesario señalar algunos elementos que se tomaron como referencia de investigaciones anteriores realizadas en este mismo campo y que resultan ser diferenciadores de estos; como se mencionó en el marco de referencia existen investigaciones sobre el mismo tema y que utilizan herramientas similares como lo son las RNA. La forma de utilización de esta herramienta fue tomada en dos enfoques; el primero es como proceso posterior a la codificación del árbol de Huffman, como una forma de reconocimiento de patrones que mejorará la aparición de estos en los códigos resultantes. El segundo enfoque trata de la utilización de una RNA de aprendizaje profundo (DNN) que produce alta precisión en los resultados de aplicaciones algorítmicas con respecto a los errores encontrados en las imágenes que componen un video a codificar. Estas investigaciones tienen en común que usan una RNA para codificar información utilizando Huffman, pero en procesos diferentes e independientes que se ejecutan de forma secuencial. La diferencia con la propuesta aquí planteada es la naturaleza de la utilización

de la RNA dentro del proceso mismo del algoritmo de codificación Huffman y no como un proceso independiente, a pesar de que en estas investigaciones también evalúan el parámetro de la complejidad del algoritmo de Huffman y adicionan la complejidad propia de la RNA, lo que es otro aporte en la propuesta ya que esta complejidad es reducida de forma implícita dentro del algoritmo. Otro factor en común es el uso de videos en las pruebas realizadas en estas investigaciones y en este trabajo sin embargo en estos trabajos se enfocan en el desempeño del algoritmo y de la RNA en dispositivos móviles, mientras que en este enfoque no se sesga hacia un tipo de dispositivo huésped, sino por el contrario en un modelo de ingeniería de software que permita la independencia de la implementación de hardware.

De este trabajo pueden desprenderse varias líneas de investigación que pueden entrar a evaluar, mejorar o replantear lo trabajado hasta este punto.

Una de estas líneas es en inteligencia computacional, donde se pueden hacer pruebas con diferentes algoritmos de entrenamiento para la RNA, distintos al Backpropagation, que permitan realizar un direccionamiento en la estructura mediante una propagación hacia adelante y hacia atrás de forma más autónoma en su recorrido y así mejorar el número de iteraciones y por consiguiente la complejidad computacional.

Este trabajo está enmarcado en los sistemas de comunicación digital, en particular en el bloque de codificación de la fuente, que permite no solo representar la información en bits, sino también realizar una compresión de los datos para mejorar el desempeño y utilización del canal; proceso que debe complementarse con una encriptación y codificación del canal, con el fin de proteger la integridad de la información durante la transmisión; por estas razones puede profundizarse en el uso de herramientas de inteligencia evolutiva para integrar y mejorar todos estos procesos en términos de tiempo y calidad de los bloques funcionales de procesamiento de información en la etapa de transmisión.

Otro trabajo posterior importante para dar validez a este mismo desarrollo es pensar en el decodificador, como siguiente paso en la puesta en funcionamiento experimental de la propuesta, lo que implica contemplar los métodos de descompresión mediante la RNA Backpropagation y cómo enviar los parámetros de la estructura junto con la información de un extremo a otro del sistema.

En el punto del receptor pueden presentarse pérdidas de información propios de las características físicas del canal de comunicaciones, por esta razón el deterioro de la información recepcionada en el decodificador puede significar mala interpretación de la misma; desde ese análisis puede pensarse en una investigación posterior en la cual se plantee la forma de integrar el proceso de corrección de errores hacia adelante utilizando la misma RNA entrenada que viene desde la codificación o que esta misma red detecte de forma inteligente los errores y los bits de redundancia para reconstruir las tramas dañadas. Con esta propuesta también puede plantearse la posibilidad de que no se utilice un método FEC con envió de bits de redundancia, por el contrario, puede encontrarse la manera que la detección de errores y la reconstrucción de la información sea realizada por la RNA, que cuenta con dichas características y así proponer un proceso de corrección de errores hacia adelante mediante redes neuronales dejando fuera los bits de redundancia, por consiguiente así mejorar las tasas de compresión ya planteadas aquí.

5. CONCLUSIONES

En esta investigación se logró demostrar que una red neuronal artificial en particular, una que utilice el algoritmo de entrenamiento Backpropagation, puede comportarse como una estructura de datos de tipo abstracto, específicamente como un grafo dirigido, para ser utilizada como generadora de códigos Huffman.

En la fase de implementación de la propuesta aquí planteada, se logró el desarrollo de los componentes pensados en la fase de diseño; estos componentes son tres; la parte del algoritmo de Huffman, el módulo de la red neuronal y por último la interfaz de entrega de los códigos.

Mediante esta arquitectura se implementó la integración de la estructura de datos tipo grafo, que es definida como el primer componente y realizada dinámicamente por la RNA, modificando iteración a iteración su topología; por último las ráfagas de información a la entrada del proceso se entregan como palabras código.

Mediante la validación y evaluación del desempeño del algoritmo propuesto; se obtuvo que las variables que intervienen en la evaluación del proceso de codificación con este algoritmo son el número de iteraciones que realiza la red neuronal para encontrar el mejor código, la complejidad computacional y la probabilidad de ocurrencia del mejor código en cierta iteración.

También se encontró que la información y su naturaleza no modifican el comportamiento del algoritmo, mientras que la variedad de la información incide en el desempeño solo durante las cuatro primeras iteraciones de la red, ya que la mejora de la tasa de compresión de las siguientes iteraciones es más evidente, debido al entrenamiento recurrente de la red.

De lo anterior también se dedujo que de la primera iteración a la 5 los cambios entre una iteración y otra son buenos, después de esta, se encuentran los mejores códigos con mayor probabilidad entre la iteración 5 y 10; después de la iteración 10 es muy poco probable que se encuentre un mejor código y por ende una mejor tasa de compresión.

También se concluye que, en el momento de utilización del algoritmo en una transmisión de información sobre un canal de comunicaciones, el usuario puede entrar a discernir sobre la ecualización de las variables de complejidad y tasa de compresión, colocándolos en una relación costo computacional sobre beneficio en tasa de compresión.

Se comprobó también que el límite teórico que Shannon planteó para algoritmos de codificación de tipo estadístico, se cumple para el algoritmo Huffman y sus modificaciones, debido a que este se basa en el proceso estadístico de clasificación de símbolos del alfabeto y por tanto no fue superado por la propuesta de este trabajo.

A pesar de la conclusión anterior, la propuesta de esta investigación obtiene mejores códigos que la forma tradicional del algoritmo de Huffman, llegando a acercarse hasta con un 10% de diferencia del límite de Shannon.

Se pudo observar que entre la iteración 5 y 10 se obtienen los mejores códigos, pero llegar a la iteración 10 implica costos de procesamiento más altos, para una tasa de compresión más constante desde la iteración 5, por tanto, se observa la máxima compresión entre un 50% a 60%.

La propuesta presentada en este trabajo se caracteriza por utilizar una RNA como una estructura de datos que genera los códigos, de esto último se concluye que la red debe entrenarse con una serie de ejemplos que van incluídos en las ráfagas de información, lo que implica una mayor cantidad de iteraciones sobre el algoritmo antes de entregar el código definitivo.

Por último, se comprobó que la desventaja de este algoritmo con respecto a otros algoritmos como el mismo Huffman Canónico, la codificación aritmética o los LZ, es que estos solo realizan una iteración de búsqueda en amplitud del código sobre la estructura; a pesar de esto la propuesta mejora considerablemente las tasas de compresión, utilizando varias iteraciones y búsqueda en amplitud realizada por la RNA.

6. REFERENCIAS

[1] H. Sklar, Bernard, FJ, “DIGITAL Fundamentals and Applications,” *Ijcnwc*, vol. 11, p. 125, 2013.

[2] J. G. Proalkis, *Digital Comunications*, 4th ed. 2008.

[3] K. Denecker, J. Van Overloop, and I. Lemahieu, “An experimental comparison of several lossless image coders for medical images,” *IEEE Conf. 1997. DCC*, p. 435, 1997.

[4] Y. Wiseman, “Compaction of RFID Devices Using Data Compression,” vol. 1, no. 3, pp. 4762–4766, 2018.

[5] N. Abramson, *Teoría de la Información y Codificación*, 5th ed. Madrid, 1981.

[6] Z. Matıas, “Compresion transparente para sistemas embebidos,” vol. 1, p. 25, 2007.

[7] B. Zhang, Z. Wang, J. Wang, and H. Wang, “Reseach on the GPRS Remote Data Compression and Transmission technology for Structure Healthy Monitoring,” *IEEE Int. Symp. Broadband Multimed. Syst. Broadcast. BMSB*, vol. 1, p. 3193, 2017.

[8] R. P. Tripathi, “Study Of Various Data Compression Techniques Used In Lossless

Compression of ECG Signals," *IEEE Commun. Mag.*, vol. 1, p. 1093, 2018.

[9] K. Sharma, "Lossless Data Compression Technique With Encryption Based Approach," *IEEE Commun. Mag.*, vol. 1, pp. 2–6, 2017.

[10] L. S. Cohen and T. Wendling, "Técnicas de diseño," *Técnicas de diseño*, vol. 1, pp. 15–18, 1998.

[11] Y. Liu and I. Engineering, "Lossless Compression of Full-Surface Solar Magnetic Field Image Based on Huffman Coding," *IEEE Commun. Mag.*, vol. 1, pp. 899–903, 2017.

[12] J. M. Hurtado, "Simulación y Análisis de Algoritmos de Compresión Empleados en un Sistema de Comunicaciones Digitales .," *Rev. Electrónica sobre Tecnol. Educ. y Soc.*, vol. 2, pp. 1–10, 2015.

[13] Á. Zavala, E. Salvador, E. Salvador, M. Linares, and E. Salvador, "Development of an Application for Data Compression by Using the Huffman Algorithm .," *IEEE J. Sel. Areas Commun.*, vol. 1, pp. 1–6, 2017.

[14] T. Mantoro, M. A. Ayu, and Y. Anggraini, "The Performance of Text File Compression Using Shannon-Fano and Huffman on Small Mobile Devices," *IEEE Access*, vol. 1, pp. 1–3, 2017.

[15] K. Coons, B. A. Maher, and K. S. Mckinley, "Optimal Huffman Tree-Height Reduction for Instruction-Level Parallelism," *Tech. Rep. TR-08-34*, vol. Department, no. 1, pp. 1–26, 2010.

[16] R. Arshad, A. Saleem, and D. Khan, "Performance comparison of Huffman Coding and Double Huffman Coding," *2016 6th Int. Conf. Innov. Comput. Technol. INTECH 2016*, pp. 361–364, 2017.

[17] D. Dath and J. V. Panicker, "Enhancing adaptive huffman coding through word by word compression for textual data," in *Proceedings of the 2017 IEEE International Conference on Communication and Signal Processing, ICCSP 2017*, 2018, vol. 2018-Janua, pp. 1048–1051.

[18] G. S. Sandeep, B. S. S. Kumar, and D. J. Deepak, "An efficient lossless compression using double Huffman minimum variance encoding technique," *Proc. 2015 Int. Conf. Appl. Theor. Comput. Commun. Technol. iCATccT 2015*, no. C, pp. 534–537, 2016.

[19] P. Walk, P. Jung, and B. Hassibi, “Short-message communication and FIR system identification using Huffman sequences,” *IEEE Int. Symp. Inf. Theory - Proc.*, no. 1, pp. 968–972, 2017.

[20] H. Wu *et al.*, “Performance comparison of Huffman Coding and Double Huffman Coding,” *IEEE Int. Symp. Inf. Theory - Proc.*, vol. 2018-Janua, no. 1, pp. 1–5, 2017.

[21] K. M. Shivkumar and N. Kashyap, “A maximization problem related to Huffman codes,” *2017 23rd Natl. Conf. Commun. NCC 2017*, vol. 1, pp. 1–6, 2017.

[22] J. R. López, F. D. De María, and F. Pérez, “Codificación de Fuente y de Canal,” *Lab. Señales y Comun.*, vol. 7, no. 2, p. 16, 2010.

[23] T. U. of Nottingham, “Introduccion a los Algoritmos Evolutivos,” *UC3M*, vol. 1, pp. 1–10, 2004.

[24] P. A. Sznajdleder, *Algoritmos a fondo con Implementacion en C y Java*, 1st ed. 2012.

[25] R. Guerequeta and A. Vallecillo, “La Complejidad en los Algoritmos,” *Técnicas de diseño*, vol. 1, p. 56, 1998.

[26] B. A. Forouzan, “Summary for Policymakers,” in *Climate Change 2013 - The Physical Science Basis*, vol. 53, no. 9, 2013, pp. 1–30.

[27] P. V. Canek and E. V. Gurovich, *Introducción a las ciencias de la computación: manual de prácticas*. UNAM, Facultad de Ciencias, 2008.

[28] J. A. Portellano Pérez, *Introducción a la computación*. Addison-Wesley, 2005.

[29] F. A. M. Gil and G. M. Quetglás, *Introducción a la programación estructurada en C*. Universitat de València, 2003.

[30] J. Bisbal Riera and J. Bisbal Riera, *Manual de algorítmica : recursividad, complejidad y diseño de algoritmos*. 2010.

[31] F. de Moya Anegón, V. Herrero Solana, and V. Guerrero Bote, “La aplicación de Redes Neuronales Artificiales (RNA): a la recuperación de la información,” *Bibliodoc Anu. Bibl. Doc. i Inf.*, pp. 147–164, 2007.

[32] H. Chen, “Machine learning for information retrieval: neural networks, symbolic learning and genetic algorithms,” *J. Am. Soc. Inf. Sci.*, vol. 46, no. 3, pp. 124–216, 1995.

[33] A. G. Blanco, “Method to approximate initial values for training lineal neural networks,” *Proc. - Electron. Robot. Automot. Mech. Conf. CERMA 2008*, pp. 443–446, 2008.

[34] A. J.Serrano, “Redes Neuronales,” *IEEE Commun. Mag.*, vol. 1, 2014.

[35] C. V. Regueiro, S. Barro, E. Sánchez, and M. Fernández-Delgado, “Modelos básicos de redes neuronales artificiales,” *Computación neuronal*. pp. 181–218, 1995.

[36] E. G. Bonilla, “Reconocimiento de caracteres mediante redes neuronales con Matlab,” *Esc. Ing.*, vol. 1, p. 213, 2005.

[37] V. Manuel *et al.*, “Red Neuronal Artificial Backpropagation aplicada al reconocimiento de dígitos hexadecimales,” *Congr. Nac. Control Automático 2017*, vol. 1, no. 2015, pp. 167–173, 2017.

[38] A. M. Pedro Larrañaga, Iñaki Inza, “Redes Neuronales,” *Dep. Ciencias la Comput. e Intel. Artif.*, vol. 1, no. 1, p. 12,17, 2011.

[39] Marco Antonio Valencia Reyes, “Algoritmo Backpropagation Conceptos y Aplicaciones,” *Inst. Politec. Investig. en Comput.*, vol. 125, p. 21, 2006.

[40] Q. Song, Y. Wu, and Y. C. Soh, “Robust adaptive gradient-descent training algorithm for recurrent neural networks in discrete time domain.,” *IEEE Trans. Neural Netw.*, vol. 19, no. 11, pp. 1841–53, 2008.

[41] A. Cruz and A. Acevedo, “Reconocimiento de voz usando redes neuronales artificiales backpropagation y coeficientes lpc,” *SEPI Telecomunicaciones ESIME IPM Mex. D.F*, vol. 1, pp. 89–99, 2008.

[42] M. A. M. Izhar, A. J. Aljohani, S. X. Ng, and L. Hanzo, “Distributed joint source coding and trellis coded modulation for symbol-based Markov sources,” *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4031–4041, 2018.

[43] K. Sharma and G. Kunal, “Lossless Data Compression Techniques and Their Performance,” *Int. Conf. Comput. Commun. Autom. Lossless*, vol. 1, pp. 256–261, 2017.

[44] B. Holland, G. R. Santhanam, P. Awadhutkar, and S. Kothari, “Statically-informed dynamic analysis tools to detect algorithmic complexity vulnerabilities,” *Proc. - 2016 IEEE 16th Int. Work. Conf. Source Code Anal. Manip. SCAM 2016*, no. 1, pp. 79–84, 2016.

[45] D. Tamir, “Delta-Huffman Coding of Unbounded Integers,” *2018 Data Compression Conf.*, vol. 1, no. November 2017, pp. 428–428, 2018.

[46] H. Wu, R. Chen, J. Wu, and Y. Huang, “A Fast Generation Algorithm of Huffman Encode Table for FPGA Implement,” *2018 8th Int. Conf. Electron. Inf. Emerg. Commun.*, pp. 21–24, 2018.

[47] L. Zhu, Y. Zhang, S. Wang, H. Yuan, S. Kwong, and H. H. S. Ip, “Convolutional Neural Network Based Synthesized View Quality Enhancement for 3D Video Coding,” *IEEE Trans. Image Process.*, vol. PP, no. XX, pp. 1–1, 2018.

[48] C. Pal, S. Pankaj, W. Akram, A. Acharyya, and D. Biswas, “Modified Huffman based compression methodology for Deep Neural Network Implementation on Resource Constrained Mobile Platforms,” *2018 IEEE Int. Symp. Circuits Syst.*, pp. 1–5, 2018.