Neural Network-Like LDPC Decoder for Mobile Applications

Fadi Jerji
Leandro Silva
Cristiano Akamine

CITE THIS ARTICLE

Jerji, Fadi; Silva, Leandro; Akamine, Cristiano; 2022. Neural Network-Like LDPC Decoder for Mobile Applications. SET INTERNATIONAL JOURNAL OF BROADCAST ENGINEERING. ISSN Print: 2446-9246 ISSN Online: 2446-9432. doi: 10.18580/setijbe.2022.1. Web Link: http://dx.doi.org/10.18580/seti-jbe.2022.1

Neural Network-Like LDPC Decoder for Mobile Applications

Fadi Jerji, Leandro Silva, and Cristiano Akamine, *Member, SET*

***Abstract*—This paper presents a low complexity iterative decoder for Low-Density Parity-Check (LDPC) codes for mobile applications using a Neural Network-like (NNL) structure and a modified Single-Layer Perceptron (SLP) training algorithm. The proposed approach allows for midrange decoding performance with a minimum gap to Shannon-limit of 3.19 dB at a frame error rate of 10^{-4} for the short frame and the code rate 13/15 of the next-generation Digital Terrestrial Television Broadcasting (DTTB) standard of the Advanced Television Systems Committee (ATSC), the "ATSC 3.0". The NNL decoder has a low decoding time, thus, it would be suitable for low power embedded systems, software-defined radio implementation tools, and software-based DTTB receptors.**

***Index Terms*—Channel coding, iterative decoding, Low-Density Parity-Check (LDPC) codes, Neural Networks.**

I. Introduction

SINCE the introduction of mobile phones in the 70s, they have become increasingly essential in our everyday lives, and with the introduction of the smartphone in the late 2000s, it started to replace many devices by combining their functionalities in one high-performance piece of hardware [1].

One of the main challenges of our modern smartphones design process is achieving a trade-off between the demand for higher processing power and multi-functionality and the cost, weight, power consumption and battery lifespan [2]. This only serves to increase the necessity for hardware and software optimization.

Most modern smartphones incorporate a variety of technologies to serve different purposes, such as the Wi-Fi, the Long Term Evolution (LTE) from the Third Generation Partnership Project (3GPP) and the under-development fifth-generation wireless technology for digital cellular networks (5G). While those technologies are vastly different in many aspects, they all share an important component, the Forward Error Correction Code (FEC) that is deployed using the Low-Density Parity-Check (LDPC) codes [3]–[5].

Another example of a technology that uses LDPC codes and is soon-to-be incorporated in smartphones is the Digital Terrestrial Television Broadcasting (DTTB) receivers, specifically the next-generation DTTB standard of the Advanced Television Systems Committee (ATSC), the "ATSC 3.0" [6].

The LDPC codes are chosen in many technologies due to their near-Shannon-limit performance but their high complexity decoders force their implementation in a specialized chipset [7].

Considering that each of those technologies has different requirements for their error correction performance and latency, several dedicated LDPC decoding chipsets have to be included in the smartphone, which increases energy requirements and price.

The remaining sections of this paper are organized as follows: the LDPC codes and the classical LDPC decoding methods in Section II, an overview of the perceptron, the Single-Layer Perceptron (SLP) and its training algorithms in Section III, the proposed Neural Networks-like (NNL) LDPC decoding method in Section IV, the mathematical complexity analysis and memory requirements of each of these decoders in Section V, the detailed results and discussions of the implemented experiments and numerical complexity calculation for each decoder in Section VI, and the conclusion in Section VII.

II. LDPC Codes

The LDPC error correction codes were introduced by Gallager in [8]. They are an attractive option for many technologies due to their near-Shannon-limit performance [7], but the LDPC code decoders are known for requiring a high processing power. Therefore, implementing these codes in software for smartphones requires highly optimized decoding algorithms, especially for the long codewords required for high-performance LDPC codes [9].

The LDPC code is represented by the notion (N,K), where N is the sum of the original information bits number K and the parity bits number P. The parity bits part $P = N - K$ is calculated and added to the original information bits, therefore generating a message that can be transmitted via a noisy channel.

An example LDPC code (7,4) defined by the matrix H is shown in (1) where the number of columns represents the total number of information bits and parity bits while the number of rows represent the number of parity bits.

The parity check matrix H can be represented by a Tanner graph representation and shown in Fig. 1 [10], where the variable nodes $V_1, V_2, \ldots, V_7$ are the values representing the message bits after passing through a noisy channel and the parity-check nodes C_1, C_2, C_3 are the calculated parity values used in error check and correction.

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

This work was supported in part by Coordination for the Improvement of Higher Education Personnel (CAPES), National Council of Technological and Scientific Development (CNPq) and MackPesquisa.

The authors are with the Postgraduate Program in Electrical and Computer Engineering (PPGEEC), Mackenzie Presbyterian University, São Paulo, Brazil.

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1

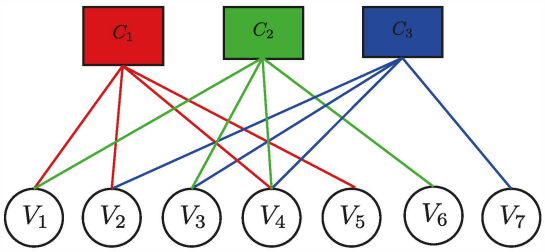

Fig. 1: Tanner Graph.

Many LDPC decoding methods were presented in the literature and generally, they can be divided into two families, high-complexity high-performance decoders and low-complexity low-performance decoders.

A. *High-complexity high-performance decoders*

The high-complexity high-performance family contains the sum-product algorithm (SPA), its simplified version the min-sum algorithm (MSA) and their variant decoding algorithms, the normalized min-sum algorithm (NMSA), the offset min-sum algorithm (OMSA) and the variable correction algorithm (VCMS) [11]–[15].

While the aforementioned decoding methods vary in performance and distance from Shannon-limit, they all share a high complexity [16]. This high complexity, when combined with a long codeword, makes a software implementation of any of them impractical, since the latency caused by the decoding process would cause a reception disturbance and make it unfeasible [17].

B. *Low-complexity Low-performance decoders*

In order to provide LDPC decoding methods with lower decoding complexity, several algorithms were introduced in the literature, such as the bit-flipping algorithm (BFA) and its variant the weighted bit-flipping algorithm (WBF) [8], [18], but their low performance makes them unattractive methods for industrial implementations.

1) Bit-Flipping decoding algorithm: The BFA algorithm proposed by Gallager [8] is the simplest LDPC decoder as it uses binary information and it does not require extensive calculation. It starts by applying (2) to calculate the parity-check nodes binary values C_{j_l} from the binary values of the variable node V_{i_l}, where i is the index of the variable node within the array of the variable nodes based on the matrix H that defines the LDPC code and j is the index of the check node within the array of the check nodes. Then for each variable node, the decoder counts the number of the unsatisfied parity-check nodes that are connected to it as in (3). Finally, the decoder flips the value of the variable node if that number was higher than a certain value X as in (4). The process is repeated until all parity-check nodes are satisfied or a maximum number of iterations is reached [8].

$$C_{j_l} = \text{XOR}_{i \in C(j)} \, V_{i_l} \tag{2}$$

$$FlipV_i = \sum_{j \in V(i)} C_{j_l} \tag{3}$$

$$V_{i_l} = \begin{cases} V_{i_l} & \text{if } FlipV_i < X \\ \text{NOT}(V_{i_l}) & \text{if } FlipV_i \geq X \end{cases} \tag{4}$$

Although this method can be easily implemented in software, its low error correction performance makes it an unattractive industrial solution in many cases [8].

2) Weighted Bit-Flipping decoding Algorithm: The WBF is an alternative LDPC decoding algorithm derived from the BFA by [18]. The decoding process starts by calculating the values of the parity-check nodes as it is done in BFA to detect any errors and terminating the decoding process if all the parity-check points were satisfied.

Otherwise, the input log likelihood ratio (LLR) the LLR_i are used to initialize the real value representation of each variable nodes V_{i_r}. Then, (5) is applied to find $|V_{i_r}|min_j$ that represents the lowest absolute real value among the variable nodes V_i that are connected to the parity-check nodes C_j. The logical representation of the parity-check nodes C_{j_l} is calculated using (6) where V_{i_l} is the logical representation of the variable nodes V_i, i and j are the indexes of the variable nodes and the check nodes respectively.

The real value E_{i_r} can be calculated using (7) then the flipping location can be determined by finding the highest E_{i_r} and flipping the logical value V_{i_l} that correspond to it using (8).

The operation is repeated until all the parity-check nodes are satisfied or the maximum number of iteration is reached.

$$|V_{i_r}|min_j = \min_{i \in C(j)} |V_{i_r}| \tag{5}$$

$$C_{j_l} = \text{XOR}_{i \in C(j)} \, V_{i_l} \tag{6}$$

$$E_{i_r} = \sum_{j \in V(i)} (2C_{j_l} - 1)|V_{i_r}|min_j \tag{7}$$

$$V_{i_l} = \begin{cases} \text{NOT}(V_{i_l}) & \text{if } E_{i_r} = \max_{i=1:N} E_{i_r} \\ V_{i_l} & otherwise \end{cases} \tag{8}$$

The WBF can perform better than the BFA in some cases, but its limitation of flipping only one bit per iteration limits its practical implementations.

III. The Single-Layer Perceptron Neural Networks

The perceptron was originally proposed by [19] and [20] as a mathematical representation of the human brain neuron and its synapses. It was developed to be able to learn the relation between its inputs $x_1, x_2, \ldots, x_n$ and its output y. According to Fig. 2, the perceptron has a bias of a fixed value of $+1$ that has a connection weight w_b. Also, each input has its own weight $w_1, w_2, \ldots, w_n$. The weights are initialized with random values at the beginning of the training and can be adjusted by the training algorithm.

Combining several perceptrons, an SLP Neural Network (NN) can be formed as shown in Fig. 3. To train the SLP, first, the network weights are randomized to break symmetry, then the training algorithm is applied as a two-step algorithm, the forward step to calculate the outputs and the second step to adjust the weights of the network to minimize the total error. For the forward step, the individual perceptron in the SLP uses

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1

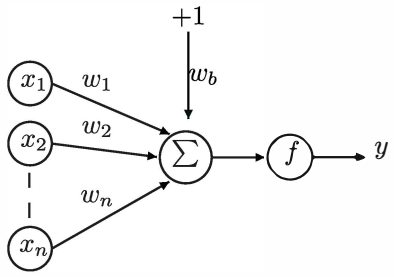

Fig. 2: The perceptron Adaline.

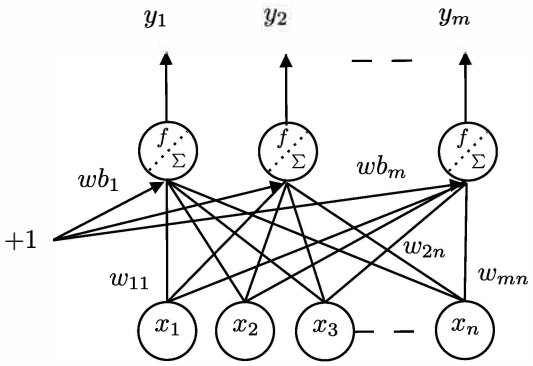

Fig. 3: Single-layer perceptron Neural Network.

a sum operation to produce its output as in (9), where x_i and yo_j are the input of the index i and the calculated output of the index j respectively.

$$yo_j = f(\sum_{i=1}^{n} x_i w_{ji} + wb_j) \tag{9}$$

The total error E is calculated using (10) where yo_j and yd_j are the calculated output and the desired output of the perceptron of the index j respectively [21].

$$E = \frac{1}{2}\sum_{j=1}^{m}(yo_j - yd_j)^2 \tag{10}$$

Then, the adjustment to the weight w_{ji}, Δw_{ji}, is calculated by the mean of the partial derivative of the total error with respect to the weight w_{ji} using (11)

$$\Delta w_{ji} = -\frac{\partial E}{\partial w_{ji}} \tag{11}$$

The final stage of the training iteration (t), is to calculate the weights for the next iteration $w_{ji}(t+1)$ applying the adjustment from the current iteration $\Delta w_{ji}(t)$ to the weights $w_{ji}(t)$ using the training rate η as in (12).

$$w_{ji}(t+1) = w_{ji}(t) + \eta \Delta w_{ji}(t) \tag{12}$$

The process is repeated until an acceptable total error or a maximum number of iterations is reached.

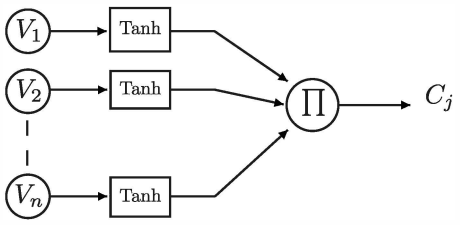

Fig. 4: The proposed XOR perceptron.

IV. The Proposed Neural Network-Like Algorithm

To provide an alternative to dedicated LDPC decoding chipsets, a configurable software implementation of the LDPC decoder for the smartphone can be used. By allowing some degradation in error correction performance, it becomes practical to create a flexible, cost-effective LDPC decoder with low power consumption.

Comparing the Tanner graph of Fig. 1 with the SLP structure in Fig. 3; the similarity can be easily spotted along with the possibility of applying the SLP NN training algorithm to decode the LDPC codes, although some modifications to the SLP and the training algorithm are required to fully match the Tanner graph.

An early attempt to implement a Multi-layer perceptron (MLP) decoder was done by [22] and revised by the same authors in [23] and while it provided a proof of concept, it was not able to deliver a stable and functional decoder for long code words due to several limitations such as the high number of multiplications and the overflow of memory register that resulted from it.

To modify the SLP structure of Fig. 3 to match the Tanner graph of Fig. 1, the bias weights should be omitted along with the activation function of each perceptron, in addition to omitting all the connections that correspond to ZERO in the matrix H and the Tanner graph. In addition to the modifications to the SLP structure, the training algorithm needs to be modified to adjust the inputs themselves instead of the weights of their connections.

The perceptron structure is not adequate for the LDPC decoder requirements since the relation between the Tanner graph inputs V and the outputs C is a logical XOR. The SLP training algorithm requires the derivation of the function XOR, which means that it should have Real numerical values as inputs and outputs instead of the Boolean XOR.

To solve the aforementioned issues, a new XOR perceptron is proposed as shown in Fig. 4.

Let real variables $\forall\ A_r,\ B_r \in \Re$, we define their logical representation A_l, B_l as the following:

$$A_l = \text{LOGICAL}(A_r) = \begin{cases} ONE & \text{if } A_r < 0 \\ ZERO & \text{if } A_r > 0 \end{cases} \tag{13}$$

Based on (13), the sign of A_r, $sgn(A_r)$, combined with its absolute value represent its probability. Thus, $A_r = 0$ is an equal probability of A_l being a logical ZERO and a logical ONE at the same time and in this case a small random value is forced. Therefore, we define:

$$\text{NOT}(A_l) = \text{LOGICAL}(-A_r)$$

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1

and

$$A_l \,\mathrm{XOR}\, B_l = \mathrm{LOGICAL}(A_r \times B_r) \tag{14}$$

As per the Tanner graph representation of the LDPC code, each check node C_j that is connected to the variable nodes $V_1, V_2, \ldots, V_n$ has a real value C_{j_r} and a logical representation C_{j_l} and each variable node has a real value V_{i_r} and a logical representation V_{i_l}. The value of C_{j_l} is calculated as in (15).

$$C_{j_l} = V_{1_l} \,\mathrm{XOR}\, V_{2_l} \,\mathrm{XOR} \ldots \mathrm{XOR}\, V_{n_l} \tag{15}$$

From (14) and (15)

$$\begin{aligned} C_{j_r} &= V_{1_r} \times V_{2_r} \times \ldots \times V_{n_r} \\ &= \prod_{i=1}^{n} V_{i_r} \end{aligned}$$

In order to avoid a variable overflow due to the limitation of real numbers microprocessors representation such as float and double float we calculate C_{j_r} as shown by Fig. 4 and (16):

$$C_{j_r} = \prod_{i=1}^{n} \mathrm{Tanh}(V_{i_r}) \tag{16}$$

As for $\forall\ V_{i_r} \in \Re$ then

$$-1 < \mathrm{Tanh}(V_{i_r}) < 1$$

and

$$-1 < \prod_{i=1}^{n} \mathrm{Tanh}(V_{i_r}) < 1$$

thus, the aforementioned limitation for the algorithm implementation is removed.
For the LDPC code (N,K), (16) becomes (17)

$$C_{j_r} = \prod_{i \in C(j)} \mathrm{Tanh}(V_{i_r}) \tag{17}$$

By modifying the SLP training algorithm for the new XOR perceptron-Tanner graph structure, we obtain (18),

$$E = \frac{1}{2} \sum_{j=1}^{P} (e_j)^2 \tag{18}$$

where E is the total error and e_j is the error for the check node C_j and is calculated using (19),

$$e_j = Z - C_{j_r} \tag{19}$$

where Z is the desired output value and is a positive real number in the range (0.0 — 1.0) representing a logical ZERO. Then the partial derivative of the total error with respect to each variable node is calculated using (20),

$$\begin{aligned} \Delta V_{i_r} &= -\frac{\partial E}{\partial V_{i_r}} \\ &= -\sum_{j \in V(i)} \left(\frac{\partial E}{\partial e_j} \times \frac{\partial e_j}{\partial C_{j_r}} \times \frac{\partial C_{j_r}}{\partial V_{i_r}}\right) \end{aligned} \tag{20}$$

where

$$\frac{\partial E}{\partial e_j} = e_j$$

and

$$\frac{\partial e_j}{\partial C_{j_r}} = -1$$

and

$$\frac{\partial C_{j_r}}{\partial V_{i_r}} = \Big[\prod_{i' \in C(j) \setminus i} \mathrm{Tanh}(V_{i'_r})\Big][\mathrm{Tanh}'(V_{i_r})]$$

The last step of the iteration would be the application of the correction to V_i using (21)

$$V_{i_r}(t+1) = V_{i_r}(t) + \eta \Delta V_{i_r}(t) \tag{21}$$

where t, $V_{i_r}(t)$ and $\Delta V_{i_r}(t)$ are the current iteration, the value of V_{i_r} during the current iteration and the calculated value of ΔV_{i_r} in the current iteration respectively and where $t+1$ and $V_{i_r}(t+1)$ are the next iteration and the value of V_{i_r} during the next iteration respectively and η is a real number that represents the correction rate. This experimental study demonstrated that the proposed decoder has its best performance when η satisfied the condition $0 < \eta < 2$.

It is important to mention that to keep the Tanh functions in (16) functioning in the linear area of the Tanh curve, the input variables V_{i_r} need to be normalized to match that area. Since the data is normalized, the NNL decoder is not sensitive to channel estimation error.

V. Complexity Analysis and Memory Requirements

To compare the complexity of the proposed NNL decoder with the low-complexity decoders, the BFA and the WBF, we calculate the number of microprocessor cycles needed for each step of each decoder and its memory footprint.

A. Complexity analysis

Let I_a, I_m, I_c, I_x, I_n, I_t, I_s and I_{abs} be the number of microprocessor cycles needed for addition, multiplication, comparison, logical XOR operation, logical NOT operation, hyperbolic tangent, signal extraction operation and absolute, respectively, nI be the total number of microprocessor cycles needed for the current step of the decoding iteration.

For the LDPC code (N,K) where O is the number of nonzero elements of the sparse matrix H of N columns and P rows, the value of O is equal to the total number of connections in the corresponding Tanner graph. The value N_{errors} is the number of erroneous bits.

For the same LDPC code (N,K), nV_i and nC_j are the number of the check nodes connected to the variable node V_i and the number of variable nodes connected to the check node Cj respectively.

1) BFA complexity: To calculate the complexity of BFA, we start by calculating the complexity of (2) that is represented by (22).

$$\begin{aligned} nI &= \sum_{j=1}^{P} nC_j \times I_x \\ &= O \times I_x \end{aligned} \tag{22}$$

The complexity of (3) is represented by (23).

$$\begin{aligned} nI &= \sum_{i=1}^{N} nV_i \times I_a \\ &= O \times I_a \end{aligned} \tag{23}$$

The complexity of (4) is represented by (24).

$$\begin{aligned} nI &= \sum_{i=1}^{N} I_c + N_{errors} \times I_n \\ &= N \times I_c + N_{errors} \times I_n \end{aligned}$$

$$\text{as } 0 \geq N_{errors} \geq N \text{ then}$$

$$N \times I_c \geq nI \geq N \times (I_c + I_n) \tag{24}$$

2) WBF complexity: To calculate the complexity of WBF, it is necessary to calculate the complexity of each step of its iteration. The complexity of (5) is represented by (25).

$$\begin{aligned} nI &= \sum_{j=1}^{P} nC_j \times (I_c + I_{abs}) \\ &= O \times (I_c + I_{abs}) \end{aligned} \tag{25}$$

The complexity of (6) is represented by (26).

$$\begin{aligned} nI &= \sum_{j=1}^{P} nC_j \times I_x \\ &= O \times I_x \end{aligned} \tag{26}$$

The complexity of (7) is represented by (27).

$$\begin{aligned} nI &= \sum_{i=1}^{N} nV_i \times (2 \times I_m + I_a) \\ &= O \times (2 \times I_m + I_a) \end{aligned} \tag{27}$$

The complexity of (8) is represented by (28).

$$nI = N \times I_x + I_n \tag{28}$$

3) NNL complexity: The complexity of the proposed NNL decoder is provided by the sum of the complexity of all its steps, starting with the calculation of the complexity of (17) that is represented by (29).

$$\begin{aligned} nI &= \sum_{j=1}^{P} [nC_j \times (I_m + I_t)] \\ &\text{Ignoring the repetitive calculations of } \mathrm{Tanh}(V_{i_r}) \\ &= I_m \times \sum_{j=1}^{P} nC_j + N \times I_t \\ &= O \times I_m + N \times I_t \end{aligned} \tag{29}$$

The complexity of (18) is represented by (30).

$$nI = P \times (2 \times I_a + I_m) + I_m \tag{30}$$

The complexity of (20) is represented by (31).

$$\begin{aligned} nI =& \sum_{i=1}^{N} \{I_a + \sum_{j \in V(i)} [I_a + (nC_j - 1) \times (I_m + I_t) \\ &+ 2 \times I_a + 3 \times I_m + I_t]\} \\ =& (N + 3 \times O) \times I_a + O \times (3 \times I_m + I_t) \\ &+ (I_m + I_t)(-O + \sum_{j=1}^{P} nC_j^2) \end{aligned} \tag{31}$$

However, since $\mathrm{Tanh}(V_{i_r})$ was pre-computed in (16), (31) becomes (32):

$$= (N + 3 \times O) \times I_a + I_m \times (2 \times O + \sum_{j=1}^{P} nC_j^2) \tag{32}$$

The complexity of (21) is represented by (33)

$$nI = N \times (I_a + I_m) \tag{33}$$

B. Memory requirements

As all methods require storing both V and C matrices that have the sizes $N \times 1$ and $P \times 1$ respectively, then we analyze the memory space that is required by each method in addition to the basic $N + P$ locations. We ignore single variables as they require a negligible memory space in comparison to the matrices used in the decoding methods.

To calculate the number of memory accesses per iteration for each decoding method, it is necessary to calculate the number of memory accesses for each step represented by nMa where Ma is the memory access per variable.

1) BFA memory requirements: The BFA requires storing the matrix $FlipV_i$ with the size $N \times 1$. Thus, the total memory requirement is: $2 \times N + P$.

The memory access of (2) is represented by (34).

$$\begin{aligned} nMa &= \sum_{j=1}^{P} Ma + \sum_{j=1}^{P} nC_j \times Ma \\ &= (P + O) \times Ma \end{aligned} \tag{34}$$

The memory access of (3) is represented by (35).

$$\begin{aligned} nMa &= \sum_{i=1}^{N} Ma + \sum_{i=1}^{N} nV_i \times Ma \\ &= (N + O) \times Ma \end{aligned} \tag{35}$$

The memory access of (4) is represented by (36).

$$\begin{aligned} nMa &= \sum_{j=1}^{N} 2 \times Ma \\ &= 2 \times N \times Ma \end{aligned} \tag{36}$$

2) WBF memory requirements: The WBF requires storing the matrices $|V_{i_r}|min_j$ and C_{j_l} with the size $P \times 1$ each. Thus, the total memory requirement is: $N + 3 \times P$.

The memory access of (5) is represented by (37).

$$\begin{aligned} nMa &= \sum_{j=1}^{P} Ma + \sum_{j=1}^{P} nC_j \times Ma \\ &= (P + O) \times Ma \end{aligned} \tag{37}$$

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1

The memory access of (6) is represented by (38).

$$\begin{aligned} nMa &= \sum_{j=1}^{P} Ma + \sum_{j=1}^{P} nC_j \times 2 \times Ma \\ &= (P + 2 \times O) \times Ma \end{aligned} \tag{38}$$

The memory access of (7) is represented by (39).

$$\begin{aligned} nMa &= \sum_{i=1}^{N} Ma + \sum_{i=1}^{N} nV_i \times (2 \times Ma) \\ &= (N + 2 \times O) \times Ma \end{aligned} \tag{39}$$

The memory access of (8) is represented by (40).

$$nMa = N \times Ma \tag{40}$$

3) NNL memory requirements: The NNL method requires storing the matrix $\mathrm{Tanh}(V_{i_r})$ with the size $N \times 1$, and the storage of the matrix ΔV_{i_r} can be omitted since only the current ΔV_{i_r} is needed, and it does not need to be stored between iterations. Thus, the total memory requirement is: $2 \times N + P$.

The memory access of (17) is represented by (41).

$$\begin{aligned} nMa &= \sum_{j=1}^{P} Ma + \sum_{j=1}^{P} [nC_j \times Ma] \\ &= (P + O) \times Ma \end{aligned} \tag{41}$$

The memory access of (18) is represented by (42).

$$\begin{aligned} nMa &= \sum_{j=1}^{P} Ma + 1 \\ &= P \times (Ma + 1) \end{aligned} \tag{42}$$

The memory access of (20) is represented by (43).

$$\begin{aligned} nMa &= \sum_{i=1}^{N} Ma + \sum_{i=1}^{N} \sum_{j \in V(i)} (nC_j + 1) \times Ma \\ &= N \times Ma + O \times Ma + Ma \times \sum_{j=1}^{P} nC_j^2 \\ &= (N + O + \sum_{j=1}^{P} nC_j^2) \times Ma \end{aligned} \tag{43}$$

The memory access of (21) is represented by (44)

$$\begin{aligned} nMa &= \sum_{i=1}^{N} Ma + \sum_{i=1}^{N} (2 \times Ma) \\ &= 3 \times N \times Ma \end{aligned} \tag{44}$$

VI. Experimental Results and Discussions

To numerically compare the proposed NNL decoder with the classical LDPC decoders, both a performance simulation and a complexity calculation are necessary.

A group of LDPC codes was selected for comparative analysis. In this paper, a case study was done using the ATSC 3.0 LDPC codes.

A. Case study: ATSC 3.0 LDPC Codes

The ATSC 3.0 has been under development as the next-generation DTTB standard by the ATSC since early 2013 [24], [25]. The ATSC 3.0 offers higher data rates, allowing higher image quality and higher robustness in comparison to the earlier ATSC 1.0 A/53 standard that was developed over a decade earlier [26].

In order for the new ATSC 3.0 standard to ensure higher robustness than its predecessors, several new techniques were adopted, such as the Bit-interleaved Coded Modulation (BICM) [27]. The BICM deploys two concatenated FEC layers with an inner LDPC code and an outer Bose-Chaudhuri-Hocquenghem (BCH) code [28]. The outer code can be replaced with a Cyclic Redundancy Check (CRC) or it can be omitted.

The ATSC 3.0 standard uses a systematic LDPC coding with two different values for its frame size N, a normal frame of 64800 bits and a short frame of 16200 bits, given that the normal frame provides a better error correction performance and the short frame a lower latency for latency-sensitive applications [29]. Ten different Code Rates (CR) from (2/15) up to (13/15) for each frame size cause a variance in the length of information part K of the LDPC message.

In order for the new ATSC 3.0 standard to achieve high robustness, two different LDPC structures are used, the irregular repeat accumulate (IRA) structure that has a high performance in medium and high CR but with the disadvantage of low performance in low CR [25], [30]. Therefore, the multi-edge type (MET) structure is used for low CR [6].

B. Performance simulation

To provide a comparative analysis, the proposed NNL was implemented and tested over an additive white Gaussian noise (AWGN) channel using quadrature phase-shift keying (QPSK) modulation [31]. Each message was decoded using a maximum of 50 iterations. An extensive computer simulation was executed to obtain the signal to noise ratio per symbol (E_s/N_0) that corresponded to the threshold of 10^{-4} frame error rate (FER) since it is always assumed that the outer coding is set to BCH. A performance of 10^{-4} FER by the LDPC inner code will guarantee an overall performance of 10^{-6} FER after applying the outer code, which is sufficient for terrestrial broadcasting services [16].

The value η was chosen to be equal to 1.0 for all code lengths and CR. For code length of 64800 and CR of (2/15 — 6/15), the normalization range and the value Z were chosen to be equal to (-2.0 — 2.0) and 0.60 respectively, and for the CR of (7/15 — 13/15) the respective equal values chosen were (-4.0 — 4.0) and 0.99. For code length of 16200 and CR of (2/15 — 4/15) and (6/15 — 8/15) the normalization range and the value Z were chosen to be equals to (-2.0 — 2.0) and 0.60 respectively, and for the CR of (5/15 and 9/15 — 13/15) the respective equal values chosen were (-4.0 — 4.0) and 0.99.

Fig. 5 demonstrates the performance of various ATSC 3.0 LDPC decoders with a code length of 16200 and CR of 13/15. In this case, the simulation was run until the total number of frames are processed regardless of the FER value, therefore

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1

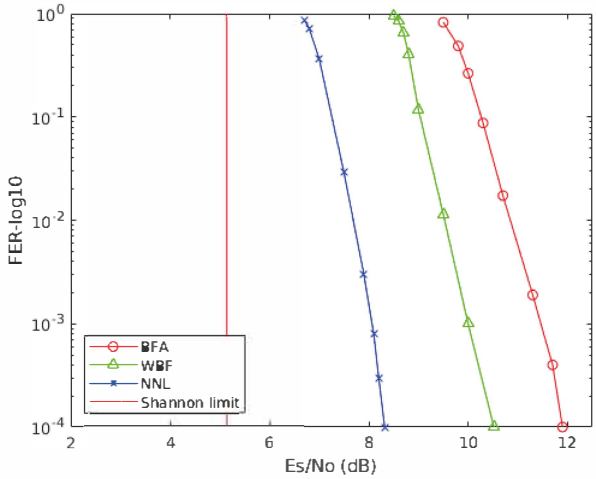

Fig. 5: ATSC 3.0 LDPC codes performance using various decoders (code length = 16200, CR = 13/15).

TABLE I: Performance comparison for ATSC 3.0 LDPC decoders (code length = 64800) (QPSK)

CR	Structure	Shannon limit (Es/N_0 ,dB)	Required Es/N_0 for FER=10^{-4} (dB)		
			NNL	WBF [32]	BFA [32]
2/15	MET	-6.92	3.81	10.32	6.80
3/15	MET	-4.94	4.90	10.32	7.90
4/15	MET	-3.47	6.02	10.34	8.41
5/15	MET	-2.26	6.70	10.36	9.31
6/15	IRA	-1.21	7.01	10.50	10.11
7/15	MET	-0.26	8.09	10.40	10.30
8/15	IRA	0.62	7.95	10.47	10.40
9/15	IRA	1.47	8.15	10.44	10.37
10/15	IRA	2.31	8.11	10.49	10.50
11/15	IRA	3.17	8.14	10.50	10.74
12/15	IRA	4.08	8.27	11.00	11.56
13/15	IRA	5.13	8.51	10.91	12.00

TABLE II: Performance comparison for ATSC 3.0 LDPC decoders (code length = 16200) (QPSK)

CR	Structure	Shannon limit (Es/N_0 ,dB)	Required Es/N_0 for FER=10^{-4} (dB)		
			NNL	WBF [32]	BFA [32]
2/15	MET	-6.92	4.04	9.21	8.70
3/15	MET	-4.94	4.90	9.13	8.16
4/15	MET	-3.47	6.32	9.20	9.03
5/15	MET	-2.26	6.50	9.17	9.10
6/15	IRA	-1.21	6.42	9.50	9.91
7/15	IRA	-0.26	7.00	10.00	9.70
8/15	IRA	0.62	6.94	10.00	10.00
9/15	IRA	1.47	7.31	9.71	10.02
10/15	IRA	2.31	7.37	10.00	10.30
11/15	IRA	3.17	7.60	10.20	10.52
12/15	IRA	4.08	7.90	10.23	10.93
13/15	IRA	5.13	8.32	10.52	11.90

it can be verified that the proposed NNL decoder does not present an error floor.

The experimental results for each coding rate are presented in Tables I and II where the NNL superior performance to both the WBF and the BFA decoding algorithm can be seen in all code rates and code lengths.

It can be seen that the limitation of the WBF of only correcting one error per iteration is clear in the results as the WBF decoder does not have a decoding performance that can achieve BER of 10^{-4} for a value of $E_s/N_0 < 9.13$ dB for any of the code rates.

The BFA decoding algorithm limitation due to its dependency on binary LLR_i values is shown in the results as well, especially at high code rates.

The proposed NNL decoding algorithm demonstrated a mid-range decoding performance getting closer to the theoretical limit in higher code rates.

C. Complexity calculation

As the hyperbolic tangent function requires one exponential function to be calculated in approximately 60 microprocessor clock cycles, while the inverse hyperbolic tangent function requires one logarithmic function thus approximately 52 processor clock cycles; most decoders use a work-around for this issue by tabling the hyperbolic tangent function to reduce the required cycles [33]. Most modern microprocessors utilize a hardware float point unit (FPU) that is capable of executing addition and multiplication in one to two cycles in addition to the logical and comparison functions in a single cycle [34]. Therefore, we can approximately calculate the number of cycles required to decode an ATSC 3.0 LDPC message of different CR using different decoders.

Considering that all the mathematical and logical operations mentioned in Section V would be executed in one processor's cycle except for the multiplication I_m that needs two cycles and the hyperbolic tangent I_t that requires 60 cycles, it is possible to calculate the number of cycles per iteration for each decoder by applying the aforementioned values in the complexity equations demonstrated in Section V. The implementation overhead caused by the various loops required to execute each step of the decoding process is estimated by counting the required operations and the number of memory accesses.

TABLE III: ATSC 3.0 LDPC decoders number of instructions and memory accesses (code length = 64800)

CR	NNL ($\times 10^6$)		WBF ($\times 10^6$)		BFA ($\times 10^6$)	
	nI	nMa	nI	nMa	nI	nMa
2/15	5.62	2.37	2.10	1.51	0.61	0.76
3/15	6.99	3.01	2.22	1.58	0.64	0.79
4/15	8.58	3.79	2.30	1.62	0.66	0.80
5/15	11.91	5.41	2.41	1.68	0.68	0.83
6/15	7.04	3.01	2.32	1.61	0.66	0.80
7/15	22.03	10.43	2.55	1.75	0.72	0.85
8/15	8.34	3.65	2.35	1.62	0.67	0.79
9/15	9.18	4.07	2.34	1.61	0.67	0.79
10/15	10.18	4.58	2.31	1.58	0.66	0.78
11/15	11.85	5.42	2.30	1.56	0.66	0.77
12/15	15.35	7.16	2.34	1.58	0.67	0.78
13/15	20.93	9.96	2.30	1.55	0.66	0.76

It can be noted that as the complexity of BFA changes based on the number of the erroneous bit, as demonstrated in (24), the presented BFA complexity calculation is done with the assumption that half of the received bits are erroneous and thus provides an average decoding time.

Tables III and IV demonstrate the complexity values calculated using the method described in Section V represented by the total number of instructions and the total number of

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1

TABLE IV: ATSC 3.0 LDPC decoders number of instructions and memory accesses (code length = 16200)

CR	NNL ($\times 10^6$)		WBF ($\times 10^6$)		BFA ($\times 10^6$)	
	nI	nMa	nI	nMa	nI	nMa
2/15	0.83	0.35	0.38	0.29	0.12	0.15
3/15	1.32	0.56	0.50	0.36	0.15	0.18
4/15	1.45	0.63	0.50	0.36	0.14	0.18
5/15	2.11	0.93	0.57	0.40	0.16	0.20
6/15	1.81	0.77	0.59	0.41	0.17	0.20
7/15	1.98	0.86	0.60	0.41	0.17	0.20
8/15	2.27	1.00	0.62	0.42	0.17	0.21
9/15	2.13	0.94	0.56	0.38	0.16	0.19
10/15	2.86	1.29	0.62	0.42	0.17	0.20
11/15	2.76	1.26	0.55	0.38	0.16	0.19
12/15	3.72	1.73	0.57	0.39	0.16	0.19
13/15	5.47	2.60	0.59	0.39	0.17	0.19

memory accesses required in a single decoding iteration of the BFA, WBF and the proposed NNL decoding algorithms for the ATSC 3.0 codes.

By analyzing the values in Tables III and IV, one can assume that the NNL decoder decoding time would be higher than the WBF decoding time, but this assumption would not be accurate even when considering the additional implementation overhead [17]. Such an assumption would be discarding two important factors, the first is the cache memory available in all practical processors that, combined with the small memory footprint of the three decoders, eliminates the need for a high number of memory accesses by storing the C and V matrices of the size P and N respectively in the cache memory thus speeding up the over-all processing [35].

The second factor is that the instructions are not necessarily executed sequentially and that currently used processors deploy the superscalar pipelining that allows for the execution speedup by benefiting from the intrinsic parallelism of the algorithm, therefore executing multiple instructions at the same time, nevertheless, any dependency in the code can limit the superscalar, especially conditional operations [36].

The proposed NNL decoding algorithm was designed with superscalar in mind by eliminating of conditional operations to avoid any branch penalties and allow for a considerable performance enhancement while the WBF and the BFA still require a high number of conditional operations relative to the total number of operations, in both cases the benefit of the superscalar pipelining is limited.

The result of both cache memory speedup and the superscalar pipelining speedup is clear in the complexity of an implementation of the three decoding methods with a clear additional speedup in the case of the proposed NNL decoder.

To calculate the decoding time for each of the decoders three states of the art mobile phones were selected, the iPhone Xs from Apple, the Galaxy S10 from Samsung and the Pixel 3XL from Google. The cycle time and memory access time is calculated based on the mobile phone specification and the values are used to calculate the decoding time for the decoders.

Although the mobile phone companies don't usually announce the specification of the processor and the memory used in their products, those specifications have been identified by tech enthusiasts. The iPhone XS deploys an A12 Bionic 2.49 GHz processor and a 64-bit single-channel 2133 MHz LPDDR4X memory while the Samsung Galaxy S10 uses a Samsung Exynos 9820 2.7 GHz processor and a Samsung K3UH7H70AM-AGCL 2133 MHz memory. The Google Pixel 3XL uses a Qualcomm Kryo 385 2.8 GHz processor and an MT53D512M64D4RQ-053_WT_E LPDDR4 1866MHz memory [37]–[39].

The values of the decoding throughput in kbps, for 50 interactions and after removing the parity bits, for the LDPC decoders are shown in Tables V and VI.

These results confirm the additional speedup that the proposed NNL decoding method gained from the superscalar pipelining in addition to the cache memory speedup that all three decoders gained.

The results for decoders throughput in kbps, for 50 interactions and after removing the parity bits, on the Samsung Galaxy S10 are shown in Fig. 6 and 7. In addition, Fig 8 and 9 demonstrate the decoders' throughput in kbps in relation to their decoding performance represented by the required E_s/N_0 for FER=10^{-4}, these figures demonstrate that even in the cases where a low code-rate BFA decoder would have a better performance than a high code-rate NNL decoder, the NNL decoder would still provide higher throughput.

Only when the E_s/N_0 is around 10 (dB), does the BFA start being more advantageous when the QPSK modulation is used, but in this case, the proposed NNL decoder might not require all 50 iterations to correct the channel-induced errors and its throughput would be higher as well. This would be a dynamic throughput value that would change according to the number of required decoding iterations.

VII. Conclusion

In this paper, we proposed an NNL low complexity LDPC decoder derived from NN and SLP training algorithm and supported with mathematical proof and computer simulation results. We demonstrated the difference in the computational complexity and memory requirements for the proposed midrange NNL, BFA, WBF decoders.

The experimental results showed that the proposed NNL outperformed both the BFA and the WBF algorithms in all of the tested code rates and code lengths of the ATSC 3.0 LDPC. In addition, the mathematical calculation along with the experimental results highlighted the speedup that the NNL decoder benefited from due to superscalar and cache memory.

The proposed NNL decoder demonstrates a low complexity and a mid-range performance; since its error correction performance is superior to the BFA and WBF while having a memory requirement and a complexity that is lower than the WBF and, in some cases, the BFA decoder. These characters makes it a very suitable decoder for software implementations for mobile applications, low-cost DTTB receivers and embedded systems.

Acknowledgment

The authors would like to thank their colleagues at the Postgraduate Program in Electrical Engineering and Computing Department and the Digital TV Research Laboratory at Mackenzie Presbyterian University.

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1

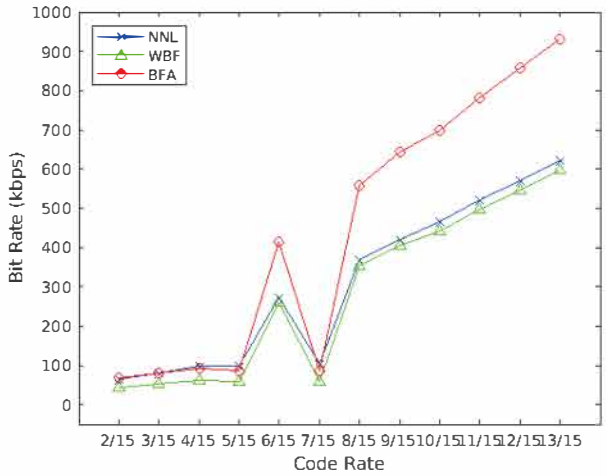

Fig. 6: ATSC 3.0 LDPC decoders throughput in kbps on Samsung Galaxy S10, (code length = 64800).

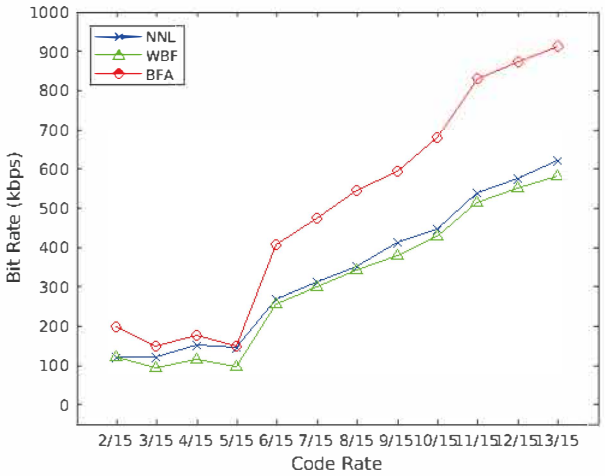

Fig. 7: ATSC 3.0 LDPC decoders throughput in kbps on Samsung Galaxy S10, (code length = 16200).

TABLE V: ATSC 3.0 LDPC decoders throughput in kbps (code length = 64800).

CR	Device	NNL	WBF	BFA
2/15	iPhone XS	58.8	40.5	62.5
	Galaxy S10	63.4	43.7	67.2
	Pixel 3XL	64.9	44.6	69.2
3/15	iPhone XS	75.3	48.8	74.9
	Galaxy S10	81.1	52.6	80.9
	Pixel 3XL	83.3	54.0	83.0
4/15	iPhone XS	90.2	56.3	86.1
	Galaxy S10	97.5	60.8	93.2
	Pixel 3XL	100.1	62.5	95.9
5/15	iPhone XS	91.3	53.3	81.1
	Galaxy S10	98.6	57.6	87.7
	Pixel 3XL	101.4	59.3	90.5
6/15	iPhone XS	254.4	243.4	386.5
	Galaxy S10	273.6	261.0	415.0
	Pixel 3XL	278.2	262.3	421.9
7/15	iPhone XS	97.0	53.1	80.4
	Galaxy S10	104.9	57.4	87.0
	Pixel 3XL	108.0	59.2	89.9
8/15	iPhone XS	340.9	327.7	519.2
	Galaxy S10	366.8	351.6	557.9
	Pixel 3XL	372.9	353.4	562.5
9/15	iPhone XS	391.4	379.7	597.9
	Galaxy S10	419.5	406.1	643.5
	Pixel 3XL	426.6	410.5	654.6
10/15	iPhone XS	434.9	413.6	649.0
	Galaxy S10	466.2	441.8	697.3
	Pixel 3XL	474.0	446.4	709.0
11/15	iPhone XS	483.4	461.8	725.1
	Galaxy S10	521.4	496.3	779.9
	Pixel 3XL	530.4	501.7	793.3
12/15	iPhone XS	527.3	508.8	797.2
	Galaxy S10	568.8	544.4	858.1
	Pixel 3XL	578.6	550.3	872.8
13/15	iPhone XS	580.4	556.8	870.5
	Galaxy S10	623.2	596.1	929.6
	Pixel 3XL	634.0	602.7	945.6

TABLE VI: ATSC 3.0 LDPC decoders throughput in kbps (code length = 16200).

CR	Device	NNL	WBF	BFA
2/15	iPhone XS	114.0	114.0	183.4
	Galaxy S10	124.1	124.1	200.9
	Pixel 3XL	125.3	123.8	200.0
3/15	iPhone XS	113.0	89.1	140.6
	Galaxy S10	121.7	95.9	150.7
	Pixel 3XL	124.6	979	153.4
4/15	iPhone XS	143.0	106.8	165.4
	Galaxy S10	153.4	115.6	179.5
	Pixel 3XL	157.2	117.9	183.6
5/15	iPhone XS	135.2	90.9	138.8
	Galaxy S10	146.5	97.7	150.7
	Pixel 3XL	149.4	100.2	154.6
6/15	iPhone XS	253.1	243.4	383.5
	Galaxy S10	269.3	258.3	408.3
	Pixel 3XL	275.4	260.6	417.2
7/15	iPhone XS	289.5	278.6	447.4
	Galaxy S10	314.2	301.3	476.3
	Pixel 3XL	319.7	302.4	483.0
8/15	iPhone XS	330.9	318.4	511.4
	Galaxy S10	351.6	344.4	544.4
	Pixel 3XL	361.4	345.1	540.9
9/15	iPhone XS	387.4	358.2	558.4
	Galaxy S10	412.7	379.7	593.3
	Pixel 3XL	424.1	386.4	612.5
10/15	iPhone XS	413.6	398.0	639.2
	Galaxy S10	448.8	430.5	680.4
	Pixel 3XL	455.3	432.7	687.3
11/15	iPhone XS	504.4	483.4	773.4
	Galaxy S10	539.6	515.6	828.7
	Pixel 3XL	551.0	524.9	831.8
12/15	iPhone XS	538.6	516.6	791.0
	Galaxy S10	575.3	550.3	872.8
	Pixel 3XL	584.2	554.7	876.5
13/15	iPhone XS	571.3	537.7	856.9
	Galaxy S10	623.2	583.4	914.1
	Pixel 3XL	633.3	587.5	924.7

REFERENCES

[1] I. Lin, B. Jeff, and I. Rickard, "Arm platform for performance and power efficiency — hardware and software perspectives," in *2016 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–5, April 2016.

[2] J. Cho, Y. Woo, S. Kim, and E. Seo, "A battery lifetime guarantee

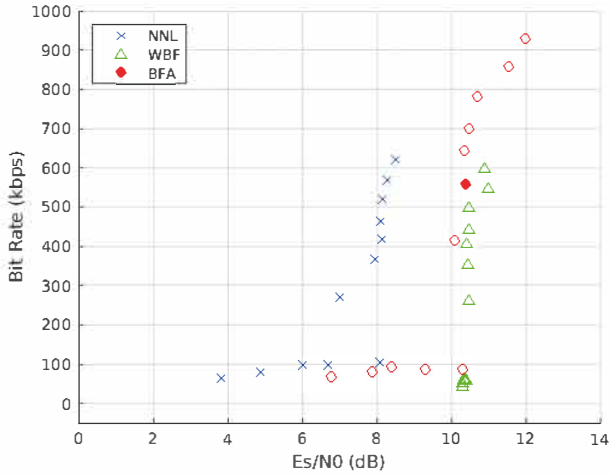

Fig. 8: ATSC 3.0 LDPC decoders throughput in kbps on Samsung Galaxy S10 in relation to their decoding performance, the required Es/N_0 for FER=10^{-4} (dB), (code length = 64800).

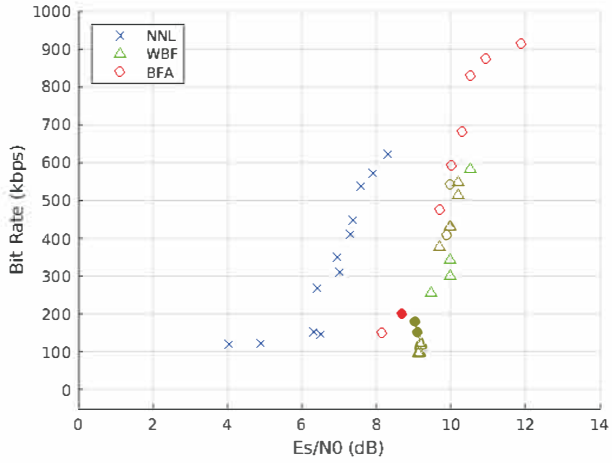

Fig. 9: ATSC 3.0 LDPC decoders throughput in kbps on Samsung Galaxy S10 in relation to their decoding performance, the required Es/N_0 for FER=10^{-4} (dB), (code length = 16200).

scheme for selective applications in smart mobile devices," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 1, pp. 155–163, 2014.

[3] IEEE-Std, "Ieee standard for information technology– local and metropolitan area networks– part 11, amendment 5," *IEEE Std 802.11n-2009*, pp. 1–565, Oct 2009.

[4] T. Richardson and S. Kudekar, "Design of low-density parity check codes for 5g new radio," *IEEE Communications Magazine*, vol. 56, pp. 28–34, MARCH 2018.

[5] G. Indumathi and D. A. Joe, "Design of optimum physical layer architecture for a high data rate lte uplink transceiver," in *2013 International Conference on Green High Performance Computing (ICGHPC)*, pp. 1–8, March 2013.

[6] K. Kim, S. Myung, S. Park, J. Lee, M. Kan, Y. Shinohara, J. Shin, and J. Kim, "Low-density parity-check codes for atsc 3.0," *IEEE Transactions on Broadcasting*, vol. 62, pp. 189–196, March 2016.

[7] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 33, pp. 457–458, Mar 1997.

[8] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, pp. 21–28, Jan 1962.

[9] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, pp. 399–431, Mar 1999.

[10] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, pp. 533–547, Sep 1981.

[11] F. R. Kschischang, B. J. Frey, and H. . Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, pp. 498–519, Feb 2001.

[12] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, pp. 673–680, May 1999.

[13] J. Chen and M. P. C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Transactions on Communications*, vol. 50, pp. 406–414, March 2002.

[14] J. Chen and M. P. C. Fossorier, "Density evolution for two improved bp-based decoding algorithms of ldpc codes," *IEEE Communications Letters*, vol. 6, pp. 208–210, May 2002.

[15] C. Chen, Y. Xu, H. Ju, D. He, W. Zhang, and Y. Zhang, "Variable correction for min-sum ldpc decoding applied in atsc3.0," in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–5, June 2018.

[16] S. Myung, S. Park, K. Kim, J. Lee, S. Kwon, and J. Kim, "Offset and normalized min-sum algorithms for atsc 3.0 ldpc decoder," *IEEE Transactions on Broadcasting*, vol. 63, pp. 734–739, Dec 2017.

[17] F. Jerji and C. Akamine, "Advanced isdb-t and atsc 3.0 ldpc codes performance and complexity comparison," *IEEE Transactions on Broadcasting*, vol. 68, no. 1, pp. 254–262, 2022.

[18] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Transactions on Information Theory*, vol. 47, pp. 2711–2736, Nov 2001.

[19] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[20] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Jun 1960.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct 1986.

[22] A. R. Karami, M. A. Attari, and H. Tavakoli, "Multi layer perceptron neural networks decoder for ldpc codes," in *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–4, Sept 2009.

[23] A. Karami and M. A. Attari, "Novel ldpc decoder via mlp neural networks," 2014.

[24] L. Fay, L. Michael, D. Gómez-Barquero, N. Ammar, and M. W. Caldwell, "An overview of the atsc 3.0 physical layer specification," *IEEE Transactions on Broadcasting*, vol. 62, pp. 159–171, March 2016.

[25] ATSC, "Physical Layer Protocol Standard, A/322:2017," standard, The Advanced Television Systems Committee, USA, June 2017.

[26] ATSC, "ATSC Digital Television Standard, A/53, Part 1:2007," standard, The Advanced Television Systems Committee, USA, Jan. 2007.

[27] L. Michael and D. Gómez-Barquero, "Bit-interleaved coded modulation (bicm) for atsc 3.0," *IEEE Transactions on Broadcasting*, vol. 62, pp. 181–188, March 2016.

[28] R. Bose and D. R. Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, no. 1, pp. 68 – 79, 1960.

[29] L. Michael and D. Gómez-Barquero, "Modulation and coding for atsc 3.0," in *2015 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, pp. 1–5, June 2015.

[30] DVB, "Part 1: DVB-S2, ETSI EN 302 307-1 V1.4.1," standard, Digital Video Broadcasting, FRANCE, Nov. 2014.

[31] A. Ghazel, E. Boutillon, J. . Danger, G. Gulak, and H. Laamari, "Design and performance analysis of a high speed awgn communication channel emulator," in *2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (IEEE Cat. No.01CH37233)*, vol. 2, pp. 374–377 vol.2, Aug 2001.

[32] F. Jerji and C. Akamine, "Gradient bit-flipping ldpc decoder for atsc 3.0," in *2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–4, Jun 2019.

[33] J. Harrison, T. Kubaska, S. Story, M. S. Labs, and I. Corporation, "The computation of transcendental functions on the ia-64 architecture," *Intel Technology Journal*, vol. 4, pp. 234–251, 1999.

[34] A. Nannarelli, "A multi-format floating-point multiplier for power-efficient operations," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pp. 351–356, Sept 2017.

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1

[35] S. Ristov and M. Gusev, "Superlinear speedup for matrix multiplication," in *Proceedings of the ITI 2012 34th International Conference on Information Technology Interfaces*, pp. 499–504, 2012.
[36] R. Khanna, S. Verma, R. Biswas, and J. Singh, "Implementation of branch delay in superscalar processors by reducing branch penalties," in *2010 IEEE 2nd International Advance Computing Conference (IACC)*, pp. 14–20, 2010.
[37] D. Yang and S. Wegner, "Apple iphone xs max teardown," 2018.
[38] M. Alarcon, D. Yang, S. Wegner, and A. Cowsky, "Samsung galaxy s10+ teardown," 2019.
[39] D. Yang and S. Wegner, "Google pixel 3 xl teardown," 2018.

FADI JERJI (S'18) received a B.S. degree in computer engineering in 2010 and an M.S. degree in electrical and computation engineering from Mackenzie Presbyterian University, São Paulo, Brazil, in 2019. He is currently pursuing a Ph.D. degree in electrical and computation engineering at Mackenzie Presbyterian University. Since 2017 he has been a post-grad Researcher with the Digital TV Research Laboratory at Mackenzie Presbyterian University.

LEANDRO A. SILVA is a Computer Engineer with a Ph.D. in Systems Engineering from the School of Engineering of the University of São Paulo, Brazil. He is currently an Adjunct Professor at the School of Computing and Informatics at Mackenzie University. He works on artificial neural networks, pattern recognition, data mining, machine learning, and big data analytics.

CRISTIANO AKAMINE received a Ph.D. degree in electrical engineering from the State University of Campinas, Brazil, in 2011. He is a Professor at Mackenzie Presbyterian University, where he is a Coordinator of the Digital TV Research Laboratory. He is a member of the Board of the Brazilian Digital Terrestrial Television Forum and Society of Brazilian Broadcast Engineers (SET). He works with the ISDB-TB broadcasting standardization and holds several patents, intellectual property licenses. He also has published numerous articles and has a Brazilian scientific grant of Productivity and Technological Development and Innovative Extension—Level 2 from the National Council of Technological and Scientific Development. He has also served as a reviewer for several periodicals and conferences and has participated as a Guest Editor in the Special Issue Point-to-Multipoint Communications and Broadcasting in 5G of IEEE Communications Magazine.

Received in 2022-09-26 | Approved in 2022-12-21

http://www.set.org.br/ijbe/ doi: 10.18580/setijbe.2022.1 Web Link: http://dx.doi.org/10.18580/setijbe.2022.1