The Syndrome Bit Flipping Algorithm for LDPC Codes

Emmanuel Boutillon, Chris Winstead, Fakhreddine Ghaffari

The Syndrome Bit Flipping Algorithm for LDPC Codes

Emmanuel Boutillon, *Senior Member, IEEE,* Chris Winstead, *Senior Member, IEEE* and Fakhreddine Ghaffari, *Senior Member, IEEE*

***Abstract*—Performance of LDPC decoders at high SNR is dominated by trapping sets that induce an error floor in the performance curve. We propose a new algorithm that resolves trapping sets and lowers the error floor. The new algorithm, called Syndrome Bit Flipping (SBF), computes the sum of adjacent parity violations at each symbol node. Bits are flipped by comparing the syndrome sum against a time-varying threshold called the decoding key. SBF is compared to other bit-flipping decoders on the Binary Symmetric Channel (BSC), and is demonstrated as a post-processing step for a Noisy Gradient Descent Bit-Flipping (NGDBF) hardware decoder. We demonstrate the post-processing method for an LDPC code defined in the 802.3an standard, and find that the frame error rate is improved by at least two orders of magnitude, even as the required iterations are reduced by 33%.**

***Keywords*—LDPC codes, bit-flipping decoders**

I. Introduction

During the past two decades, Low Density Parity Check (LDPC) codes have emerged as a crucial component of many communication standards, and are increasingly important in memory technologies. One of the main challenges for LDPC decoders is to reduce or eliminate the error floor. To achieve this, many methods of post-processing have been developed, which can reduce the floor by up to a few orders of magnitude. In this paper, we consider a method of post-processing that is especially suited to bit-flipping algorithms. The new algorithm, called Syndrome Bit Flipping (SBF), ignores channel information and considers only the parity-check syndrome states for the final iterations of bit flipping. Algorithm diversity is obtained by varying a bit-flipping threshold in each post-processing iteration.

In order to motivate the syndrome-only strategy, we first consider an analogy to compressed sensing. We then show that a threshold sequence, which call the *decoding key*, can be chosen to resolve dominant trapping set errors that are commonly associated with LDPC error floors. This method of post-processing lowers the error floor in bit-flipping decoders, and costs minimal hardware overhead since it only requires masking part of the existing operations in bit-flipping decoders. There is some cost of extra iterations needed for post-processing, however our simulations show that post-processing allows using fewer iterations of the primary decoder, so the total iterations are actually decreased while still obtaining better performance.

The remainder of the paper is organized as follows: Sec. II-B describes the motivation, notation, and steps of the proposed syndrome-only decoding algorithm. Sec. III presents a detailed analysis of the algorithm for common trapping sets that are known to be dominant contributors to error floors in standard LDPC codes. In Sec. IV we present simulation results of the algorithm as a post-processing solution for Noisy Gradient Descent Bit Flipping (NGDBF) [1], Probabilistic Gradient Descent Bit Flipping (PGDBF) [2], [3] and their variant algorithms [4] simulated on additive white Gaussian noise (AWGN) and Binary Symmetric Channel (BSC), respectively. Finally, Sec. V offers discussion and conclusions.

Notation: In the sequel, binary variables and loop indexes are represented by lowercase letters. Integer variables are represented by capital letters, integer algorithm parameters by Greek letters, and matrices by bold-face capital letters.

II. Syndrome Based decoding algorithm

A. Compressed Sensing Analogy

In the classical theory of block codes, it is well known that when a single error is present in a received data frame, the location of the error corresponds to a unique syndrome pattern that can be obtained via a simple binary calculation. For some codes, this concept can be extended to correct multiple errors. Such a concept is not readily available for modern LDPC codes, where there can be a larger number of initial errors in the received frame. We can, however, make a weaker argument based on compressed sensing: when the number of errors is small relative to the code's frame length, then the syndrome pattern most likely corresponds to a unique error pattern. In compressed sensing, a vector $\vec{v}$ of dimension n with only a few non zero components, say t with $t \ll n$, can be compressed into a vector $\vec{w}$ of size $m = t(1+\epsilon)$, with $\epsilon \ll 1$, by multiplying $\vec{v}$ with a random matrix $\mathbf{R}$ of size (m, n): $\vec{w} = \mathbf{R}\vec{v}$.

Compressed sensing is generally performed with real or complex numbers. Nevertheless, the theory is applicable to coding theory where operations are done over a Galois Field. Let us consider the IEEE 802.3an LDPC code defined for the 10 Gbit/s Ethernet Standard [5]. This LDPC code is defined by a parity-check matrix $\mathbf{H}$ with size 2048×384. It is a regular code with check degree $d_c = 32$ and variable degree $d_v = 6$. By definition, any codeword $\vec{c}$ verifies $\mathbf{H}\vec{c} = \vec{0}$. Now let us assume that the codeword is received through a

E. Boutillon is with Lab-STICC, UMR 6285 CNRS, Université Bretagne Sud, 56100 Lorient, France. Email: emmanuel.boutillon@univ-ubs.fr

C. Winstead is with the Department of Electrical and Computer Engineering, Utah State University, Logan, UT, 84322 USA. Email: chris.winstead@usu.edu

F. Ghaffari is with the ETIS Lab, ENSEA, CY Cergy Paris University, France. Email: fakhreddine.ghaffari@cyu.fr

Manuscript received February 2023

BSC. The received symbols are $\vec{d} = \vec{c} \oplus \vec{e}$ where $\vec{e}$ is a binary vector of length n, with t non null components (i.e. t errors of transmission). Thus, the syndrome $\vec{s} = \mathbf{H}\vec{d} \mod 2$ is equal to $\vec{s} = \mathbf{H}(\vec{c} \oplus \vec{e}) = \mathbf{H}\vec{e}$ and depends only on the error pattern $\vec{e}$.

To give a more exact example of this analogy, suppose a given received codeword of the (2048, 1664) 802.3 LDPC code with $t = 20$ errors and an associated syndrome with 60 non-null components. The number N_e of possible error patterns with $t = 20$ is

$$N_e = \binom{20}{2048} = 6.31 \times 10^{47}, \tag{1}$$

while the number of syndromes N_s with 60 non-null values among $m = 384$ is equal to

$$N_s = \binom{60}{384} = 1.01 \times 10^{71}. \tag{2}$$

Since $N_s \gg N_e$, the set of error patterns is quite sparse relative to the set of syndrome patterns. Then one can consider, as in the case of compressed sensing, that the knowledge of s is enough to characterize entirely the error pattern e when the number of errors t is small enough (say lower than 20 for example).

This observation motivates us to search for new types of decoding algorithms based only on the syndrome, ignoring the received samples $\vec{y}$ in the decoding process, except for the initial computation of the syndrome $\vec{s}$. In the case of post-processing, we suppose that a soft-information decoder, operating in the error-floor region, is applied to the received samples $\vec{y}$, and produces a vector $\vec{d}$ of binary decisions. In the error floor region, the decoder's output should have at most a relatively small number of errors corresponding to some absorbing set. We propose that these residual errors should be correctable using a syndrome-only calculation.

B. The SBF Algorithm

The Syndrome Based Flipping algorithm can be defined as a variant of the Gradient Descent Bit Flipping (GDBF) algorithm, with two fundamental differences. First, the channel information is no longer taken into consideration. Second, the flipping decision is based on a predefined threshold $\theta(\ell)$ that depends on the iteration number ℓ only. In principle, SBF can be used for post-processing with any LDPC decoding algorithm; in this paper we consider SBF in relation to GDBF due to its close similarity and straightforward hardware adaptation.

To describe the algorithm's steps, we first briefly restate the main points of the NGDBF algorithm [1]. Given received sample vector $\vec{y}$, a symbol-wise energy function is computed. The NGDBF energy function can be expressed as

$$E_k^{(\text{NGDBF})} = -\alpha \hat{x}_k y_k + \sum_{j \in \mathcal{M}_k} s_j + q_k, \tag{3}$$

where $\hat{x}_k \in \{+1, -1\}$ is the bipolar decision at iteration ℓ, $\mathcal{M}_k$ is the set of parity-checks adjacent to $\hat{x}_k$, $s_j \in \{0, 1\}$ are the corresponding binary parity syndrome values, q_k is an artificial noise perturbation, and α is a scale factor. The decision $\hat{x}_k$ is flipped if E_k exceeds a flipping threshold θ.

In simplest terms, the SBF algorithm modifies the energy function by considering only the adjacent parity information:

$$E_k^{(\text{SBF})} = \sum_{j \in \mathcal{M}_k} s_j. \tag{4}$$

$E_k^{(\text{SBF})}$ is equal to the number of non-satisfied checks adjacent to variable x_k. The corresponding bit is flipped if $E_k > \theta(\ell)$, where $0 \leq \theta(\ell) \leq \max(d_v)$ (similar to the original parallel M-GDBF algorithm [6]). The precise steps of the SBF algorithm are detailed in Algorithm 1. When SBF is used for post-processing in bit-flipping algorithms, the hardware cost is quite low. This is demonstrated in Fig. 1 for the case of NGDBF, where a global mode-select signal is routed to mask the channel sample and noise by forcing $q_k \coloneqq 0$ and $y_k \coloneqq 0$.

Algorithm 1 Syndrome Based algorithm

Input data:
Noisy binary codeword $\vec{d}$ and parity check matrix $\mathbf{H}$.
List of S_n sequences of thresholds, $\{\Theta_k\}_{k=1,\dots S_n}$ with $\Theta_k = \{\theta_k(\ell)\}_{\ell=1\dots N_k}$ where N_k is the length of the k^{th} sequence.
Initialization:
Compute $\vec{s}_0 = \mathbf{H}\vec{d} \mod 2$; Set $k = 0$;
Algorithm:
while $k < S_n$ **and** $\vec{s} \neq \vec{0}$ **do**
  $\ell = 0$; $k = k + 1$; $\vec{s} = \vec{s}_0$; $\vec{e} = \vec{0}$;
  while $\ell < N_k$ **and** $\vec{s} \neq \vec{0}$ **do**
    $\ell = \ell + 1$;
    $\vec{E} = \mathbf{H}^T\vec{s}$;
    for $i = 1$ **to** N **do**
      if $\vec{E}(i) > \theta_k(\ell)$ **then**
        $\vec{e}(i) = 1 - \vec{e}(i)$;
      end if
    end for
    $\vec{s} = \mathbf{H}\vec{e} \mod 2$;
  end while
end while
if $\vec{s} \neq \vec{0}$ **then**
  $\vec{e} = \vec{0}$
end if

return $\vec{z} = \vec{d} + \vec{e}$

III. Trapping Set Analysis

It is now well known that error floors of LDPC codes are affected by cycle subgraphs known as absorbing sets or trapping sets, and for BP-based decoders there are many approaches for mitigating them, e.g. [7]. It was previously shown that decoder diversity — dynamic algorithm rules that change from one iteration to the next — can be exploited to resolve trapping set errors and improve performance of low-complexity decoding algorithms, e.g. in the framework of Finite Alphabet Iterative Decoders (FAID) [8], [9]. In this

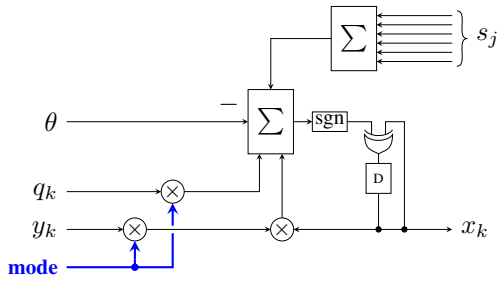

Fig. 1. Modified symbol node with dual-function capability for both NGDBF and SBF. The added "mode" input is a binary signal that selects NGDBF when mode $= 1$, and SBF when mode $= 0$.

section, we examine decoder diversity in SBF by way of the dynamic flipping threshold. We show that a sequence of threshold values can be devised to incrementally resolve known dominant error patterns due to trapping sets.

A. Isolated trapping sets

We first consider trapping sets in isolation, ignoring their neighborhoods within a larger code. For a each particular trapping set, we obtain a specific threshold sequence that corrects all symbol nodes within the set, independent of the initial condition. We refer to such a sequence as a *decoding key* for the trapping set. We do not offer a proof that such sequences exist for all trapping sets. Instead they are discovered through exhaustive search, and sequences are reported in supplemental data for 29 different trapping set topologies [10].

As a first example, a (3,3) trapping set is shown in Fig. 2 for three different error configurations. If all bits are initially in error (Fig. 2.a), then the local parity sum will be $E = 1$ at each of the symbol nodes in the set. If the first flipping threshold is set to $\theta(1) = 0$, then all erroneous bits are flipped.

If only two symbols are in error in the (3,3) trapping set (Fig. 2.b), then all symbols will have a parity sum $E = 2$. If the second threshold is $\theta(2) = 1$, then all bits are flipped, resulting in a single-error condition where the correct symbols have $E = 1$ and the erroneous symbol has $E = 3$. To correct this final case (Fig. 2.c), the third threshold is adjusted to $\theta(3) = 2$, so that the erroneous bit is corrected.

A more generalized analysis is obtained by analyzing the error state transition diagram for the trapping set. One example is shown in Fig. 3, where the states represent error patterns within the trapping set subgraph, with 000 being the error-free state. The edges in Fig. 3 are labeled with the value of $\theta(\ell)$ that will induce a transition from one error pattern to another. From the state transition diagram, we deduce a set of Θ sequences guaranteed to reach state 000 regardless of the initial state. In this example, there are six sequences of length three which guarantee correction regardless of the initial state: $\Theta_1 = \{0, 1, 1\}$, $\Theta_2 = \{0, 1, 2\}$, $\Theta_3 = \{1, 1, 0\}$, $\Theta_3 = \{1, 2, 0\}$, $\Theta_5 = \{2, 0, 1\}$ and $\Theta_6 = \{2, 0, 2\}$.

Multiple trapping sets can be resolved if the dominant trapping set graphs are known. For each known trapping set, the transition analysis is performed to obtain decoding keys that resolve each trapping set type. The keys are concatenated to produce a complete key for the code. Since trapping sets are small graphs, it is straightforward to enumerate all paths, increasing the path length until the shortest decoding keys are discovered. This method was used to find solutions for 29 trapping sets selected from a public dataset [11], and the resulting decoding keys are provided as supplemental data.

B. Heuristics for non-isolated trapping sets

We say that a trapping set is non-isolated if the SBF algorithm causes error propagation in the set's neighborhood. Errors propagate outside the trapping set neighborhood whenever $\theta(\ell) = 0$. The problem is illustrated in Fig. 4, which shows that the neighbors of an erroneous symbol node all have $E = 1$, so they will be flipped to erroneous states whenever $\theta = 0$. If a θ sequence contains repeated zeros, then errors tend to propagate to a large neighborhood.

Heuristic 1: Whenever $\theta(\ell) = 0$, $\theta(\ell + 1) > 0$ and $\theta(\ell + 2) > 0$. *Rationale:* Simulations indicate that in most instances, propagated errors are corrected if the zero threshold is followed by one or more non-zero threshold values in the decoding key.

Heuristic 2: Error propagation is suppressed by inserting $\theta = d_v - 1$ and $\theta = d_v - 2$ into the decoding key. *Rationale:* A solitary error (i.e. an erroneous symbol with no adjacent errors) may be produced as a consequence of error propagation, and is corrected when $\theta = d_v - 1$. Similarly, an adjacent pair of erroneous symbols is corrected when $\theta = d_v - 2$.

Heuristic 3: SBF is only applied for bits with "unreliable" channel samples, $|y| \leq \gamma$ for some threshold γ. The choice of γ is determined empirically by simulation. *Rationale:* This heuristic bounds the neighborhood in which errors may propagate, thereby improving the probability that all errors can be resolved.

IV. Performance Evaluation

A. SBF Performance on BSC

We first evaluate the SBF algorithm on the BSC as a standalone decoding algorithm, and evaluate its performance in comparison to other bit-flipping algorithms for the IEEE 802.3an LDPC code. The error floor for this code is dominated by an (8,8) trapping set pattern. The decoding key for this simulation was prepared using Heuristics 1 and 2, and is provided online as a supplemental file.

The simulation results are shown in Fig. 5. At frame error rates above 10^{-6}, standalone SBF performance is somewhat worse than PGDBF or BP. The improvement of SBF is seen in the error floor region, where the frame error rate approaches 10^{-8}. In this performance domain, SBF performs better than either PGDBF or BP, likely because of its resistance to dominant trapping set errors that affect the other algorithms.

B. SBF as Post-Processing on AWGN

SBF was implemented as a post-processing option for the NGDBF algorithm, and tested on two codes. Decoding keys were generated applying Heuristics 1 and 2 as described in Section III. The first code is the IEEE 802.3an LDPC code,

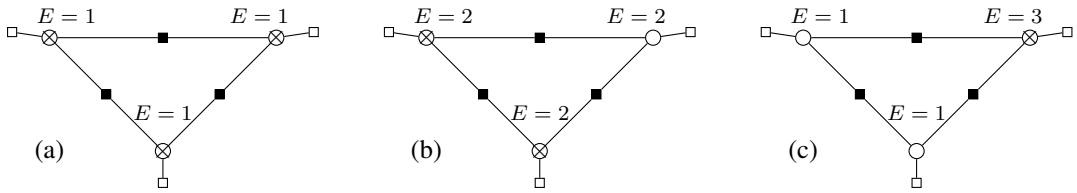

Fig. 2. Example of trapping set correction under dynamic threshold flipping. Erroneous symbol nodes are indicated by ⊗; degree-two check nodes appear as solid boxes, and degree-one check nodes as unfilled boxes. At each symbol node, the total local parity sum E is indicated. Three states are shown: (a) the initial state with three errors, (b) two errors, and (c) one error.

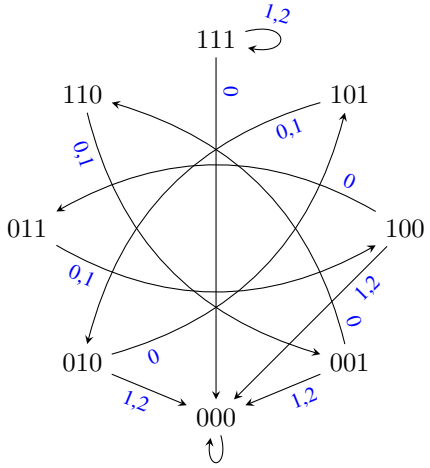

Fig. 3. Error states of the (3,3) trapping set. The edges indicate transitions of the SBF algorithm, and are labeled with values of $\theta(\ell)$ that induce the corresponding transition.

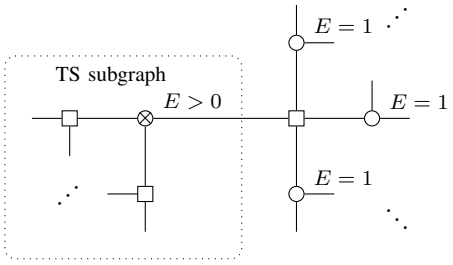

Fig. 4. For an erroneous symbol node within a trapping set subgraph, adjacent correct symbol nodes have $E = 1$. These bits are flipped whenever $\theta = 0$, causing error propagation.

simulated using FPGA emulation of a 5-bit quantized AWGN channel. The results are shown in Fig. 6. The figure includes quantized BP (5-bit) and unquantized BP, using data from Zhang et al. [12]. NGDBF was found to reach an error floor similar to quantized BP at an FER of 10^{-6}. With SBF post-processing, the error floor is lowered to about 10^{-8}, similar to unquantized BP.

The results also show that the total iterations can be significantly reduced when using SBF, requiring a maximum of four hundred iterations for NGDBF+SBF, whereas NGDBF in

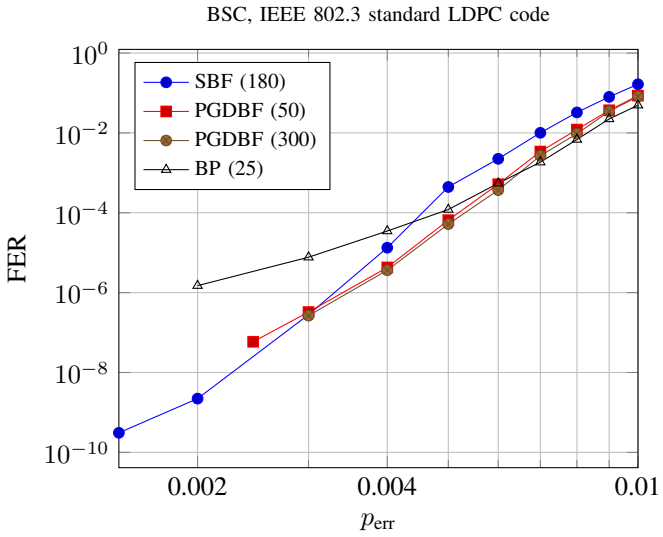

Fig. 5. Test results obtained from simulation of the rate 0.8413 802.3an code on the BSC, compared to the recent PGDBF algorithm and the standard belief propagation (BP) algorithm.

isolation must allow at least 600 iterations to achieve adequate performance on this code, up to 1000 iterations for best performance [13]. By lowering the maximum iterations from 600 to 400, the decoder's worst-case throughput is improved by 33% in addition to the improved error floor.

The final result in Fig. 6 shows results for NGDBF with SBF post-processing where Heuristic 3 is applied. These results were obtained by capturing error frames from the hardware emulation. Those frames were then decoded again in software with NGDBF+SBF applying all three heuristics.

The second code simulated was a larger code with $d_v = 5$ and $N = 10240$, with results shown in Fig. 7. This code reaches a very high error floor with NGDBF, but the floor is suppressed below an FER of 10^{-6} when using NGDBF+SBF.

V. Conclusion

We presented a new low-complexity LDPC decoding algorithm suitable for post-processing to lower the error floor. The method has a very simple implementation because it mainly requires disabling a portion of the symbol node calculations. Some overhead is also needed to distribute a global integer

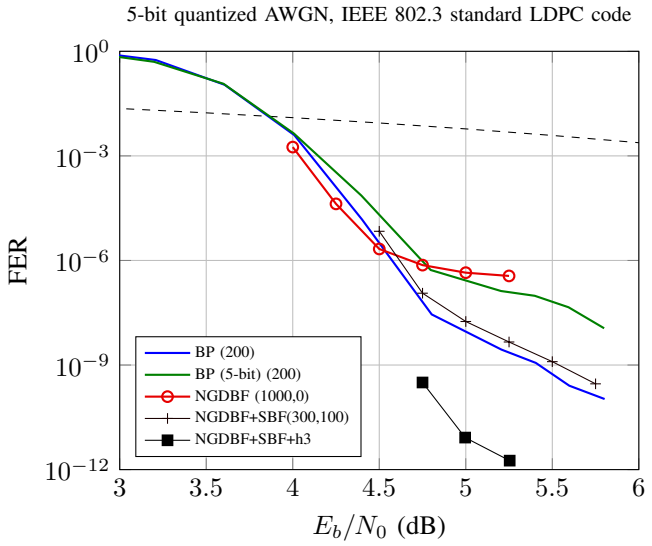

Fig. 6. Test results obtained from an FPGA implementation of NGDBF with post-processing for the 802.3an code. The NGDBF+SBF results use 300 iterations of NGDBF followed by 100 iterations of SBF. The NGDBF+SBF+h3 result indicates that Heuristic 3 was applied.

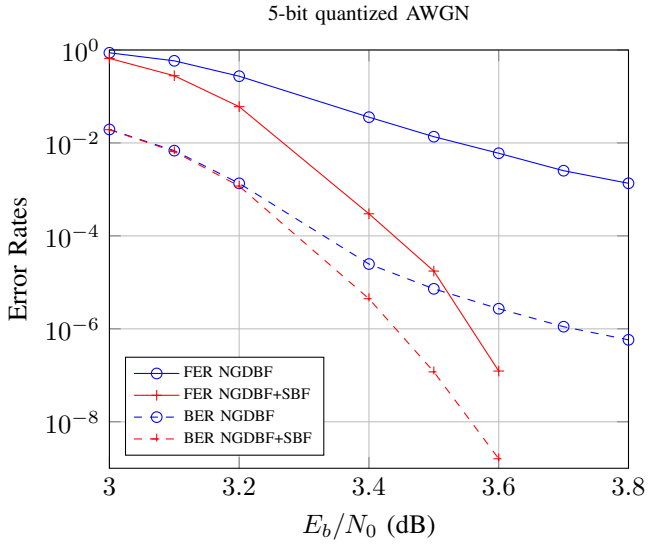

Fig. 7. Simulation results of NGDBF with post-processing for a regular $(5, 20)$ LDPC code of length $N = 10240$. NGDBF was used for up to 1000 iterations, followed by 11 post-processing iterations with SBF.

threshold value that changes in each iteration. A procedure was given for optimizing the threshold sequence in order to resolve errors on known dominant trapping sets in the code.

APPENDIX

The set of 17 decoding key sequences $S_{17} = \{\Theta_k\}_{k=1,2,\ldots,17}$ used for the results in Fig. 5 are given below. They are sorted in descending order, for clarity. In a real application, the order of execution of each decoding key should be modified to perform first the decoding key sequences that correct the highest number of remaining errors. This smart ordered reduced the average iteration before finding a codeword.

$\Theta_1 = \{5534324\};$
$\Theta_2 = \{5454543332\};$
$\Theta_3 = \{544442543333\};$
$\Theta_4 = \{5443332333233233233233233143323323\};$
$\Theta_5 = \{544233242332\};$
$\Theta_6 = \{5433343233333\};$
$\Theta_7 = \{5435432433433333323\};$
$\Theta_8 = \{534434333333\};$
$\Theta_9 = \{53343333\};$
$\Theta_{10} = \{45433333233\};$
$\Theta_{11} = \{443433322332333\};$
$\Theta_{12} = \{44233\};$
$\Theta_{13} = \{433334233\};$
$\Theta_{14} = \{354333333\};$
$\Theta_{15} = \{34433324332\};$
$\Theta_{16} = \{333332433\};$
$\Theta_{17} = \{323244423\}.$

REFERENCES

[1] G. Sundararajan, C. Winstead, and E. Boutillon, "Noisy gradient descent bit-flip decoding for LDPC codes," *IEEE Trans. Comm.*, vol. 62, no. 10, pp. 3385–3400, Oct 2014.

[2] O. A. Rasheed, P. Ivanis, and B. Vasic, "Fault-tolerant probabilistic gradient-descent bit flipping decoder," *IEEE Comm. Lett.*, vol. 18, no. 9, pp. 1487–1490, Sep. 2014.

[3] K. Le, F. Ghaffari, L. Kessal, D. Declercq, E. Boutillon, C. Winstead, and B. Vasic, "A probabilistic parallel bit-flipping decoder for low-density parity-check codes," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 403–416, 2019.

[4] V. Savin, "Gradient descent bit-flipping decoding with momentum," in *Int'l Symp. on Topics in Coding (ISTC)*, 2021, pp. 1–5.

[5] "IEEE Standard for Ethernet," *IEEE Std. 802.3-2015*, 2016.

[6] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," *IEEE Trans. Comm.*, vol. 58, no. 6, pp. 1610–1614, June 2010.

[7] Y. Han and W. E. Ryan, "Low-floor decoders for LDPC codes," *IEEE Trans. Comm.*, vol. 57, no. 6, pp. 1663–1673, June 2009.

[8] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders part I: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Trans. Comm.*, October 2013.

[9] D. Declercq, B. Vasic, S. K. Planjery, and E. Li, "Finite alphabet iterative decoders part II: Towards guaranteed error correction of LDPC codes via iterative decoder diversity," *IEEE Trans. Comm.*, October 2013.

[10] "The Syndrome Bit Flipping Algorithm for LDPC Codes, future IEEE Supplementary Data with a number to be defined later," https://cloud.etis-lab.fr/index.php/s/oJ8NEGmJe7eqjEH, accessed: 2023-02-23.

[11] "Trapping set ontology," http://www2.engr.arizona.edu/~vasiclab/Projects/CodingTheory/TrappingSetOntology.html, accessed: 2023-02-18.

[12] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. J. Wainwright, "Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices," *IEEE Trans. on Comm.*, vol. 57, no. 11, pp. 3258–3268, Nov 2009.

[13] G. Sundararajan and C. Winstead, "ASIC design of a noisy gradient descent bit flip decoder for 10GBASE-T ethernet standard," arXiv:1608.06272, 2016.