entropy

MDPI

Article

Suspicion Distillation Gradient Descent Bit-Flipping Algorithm

Predrag Ivaniš [1,*], Srdjan Brkić [1] and Bane Vasić [2]

[1] School of Electrical Engineering, University of Belgrade, 11000 Belgrade, Serbia; srdjan.brkic@etf.rs
[2] Department of ECE, University of Arizona, Tucson, AZ 85721, USA; vasic@ece.arizona.edu
* Correspondence: predrag.ivanis@etf.bg.ac.rs

Abstract: We propose a novel variant of the gradient descent bit-flipping (GDBF) algorithm for decoding low-density parity-check (LDPC) codes over the binary symmetric channel. The new bit-flipping rule is based on the reliability information passed from neighboring nodes in the corresponding Tanner graph. The name *SuspicionDistillation* reflects the main feature of the algorithm—that in every iteration, we assign a level of suspicion to each variable node about its current bit value. The level of suspicion of a variable node is used to decide whether the corresponding bit will be flipped. In addition, in each iteration, we determine the number of satisfied and unsatisfied checks that connect a suspicious node with other suspicious variable nodes. In this way, in the course of iteration, we "distill" such suspicious bits and flip them. The deterministic nature of the proposed algorithm results in a low-complexity implementation, as the bit-flipping rule can be obtained by modifying the original GDBF rule by using basic logic gates, and the modification is not applied in all decoding iterations. Furthermore, we present a more general framework based on deterministic re-initialization of the decoder input. The performance of the resulting algorithm is analyzed for the codes with various code lengths, and significant performance improvements are observed compared to the state-of-the-art hard-decision-decoding algorithms.

Keywords: bit-flipping algorithm; decoder re-initializations; gradient descent; iterative decoding; low-density parity-check codes

Citation: Ivanis, P.; Brkić, S.; Vasić, B. Suspicion Distillation Gradient Descent Bit-Flipping Algorithm. *Entropy* **2022**, *24*, 558. https://doi.org/10.3390/e24040558

Academic Editor: Syed A. Jafar

Received: 9 March 2022
Accepted: 14 April 2022
Published: 15 April 2022

1. Introduction

Low-density parity-check (LDPC) codes [1] are powerful error correction codes that achieve performance near the Shannon limit with low decoding complexity [2]. These codes have been adopted in the fifth-generation standard for broadband cellular networks (5G NR) [3], digital video broadcasting standards for satellite communications (DVB-S2 and DVB-S2X, [4]), communication standards for Wi-Fi and WiMax networks [5,6], and passive optical networks [7].

The decoding algorithms of practical interest are typically based on a message-passing approach. The decoding procedures of LDPC codes are described by a bipartite graph—known as a Tanner graph—comprising two sets of nodes—variable nodes (VN) and check nodes (CN), and a set of edges between them. During decoding, messages are iteratively exchanged between VNs and CNs connected by an edge, i.e., the messages are sent to the neighbors. A prime example of a message-passing decoder is the belief propagation (BP) algorithm [1]. While the BP decoder exhibits excellent performance, this state-of-the-art algorithm requires complex computations in variable and check nodes. Consequently, its approximate variants, such as min-sum [8] and offset min-sum [9], are typically used in contemporary communication systems. All of these variants use "soft" (real-valued) messages and higher numerical precision of these messages to improve error correction capability at the cost of increasing hardware complexity and reducing throughput.

The BP-based decoding can be considered an iterative process of Bayesian inference and the messages are the representations of the corresponding probabilities. Although optimal for extremely long codewords, the performance of short and medium-length LDPC

codes is far from optimal, due to increased error floors and the large number of decoding iterations needed to reach the desired decoding reliability. Given the fact that decoding latency is proportional to the codeword length, the aforementioned drawbacks limit the usefulness of LDPC codes in delay-sensitive applications, such as the URLLC (Ultra-Reliable and Low-Latency Communications) 5G radio service category [10]. Furthermore, the decoding complexity of message-passing algorithms increases significantly with the the number of edges that connects VNs and CNs. To mitigate the drawbacks of the BP decoder, finite alphabet iterative decoders (FAIDs) have been proposed in [11], and it is based on the idea to convey the local structure of the Tanner graph by employing a small number of additional bits in the messages.

On the other hand, bit flipping (BF) is the simplest iterative decoding algorithm that operates with one-bit messages and can provide very high throughputs when only hard decisions are available at the channel output. The BF algorithm has lower complexity when compared with the simplest hard-decision message-passing algorithm (Gallager-B), as there is no passing of the messages between the nodes in the bipartite graph (a comprehensive complexity analysis of the hard-decision decoders can be found in [12]). Although the implementation complexity of the BF algorithm is low, its performance is inferior when compared to the algorithms passing soft messages. The guaranteed error correction capability of the BF decoder is dominantly determined by two parameters of the bipartite graph: the girth and variable node degree (number of parity checks connected to the VN). As the girth grows logarithmically with the codeword length [13], the error correction capability of the BF-based decoders is poor for the short codes with small VN degrees.

Recent research attempts are aimed at providing decoding algorithms with a complexity comparable to the simple bit-flipping type of decoders but with a performance close to soft message-passing decoders. It has been shown that the gradient descent bit-flipping (GDBF) algorithm [14], together with the recently proposed improvements, has the potential to meet these requirements.

In this paper, we identify the drawbacks of the GDBF algorithm responsible for failing to correct some low-weight error patterns. We propose modifications that address these drawbacks, and thus improve the performance and convergence speed, even for codes with short blocklength and a small VN degree.

1.1. Related Work

The main idea of the GDBF algorithm [14] is that a non-linear objective function is defined for every VN in every iteration. It is calculated based on the received word from the channel, the current bit estimate for this VN, and the number of unsatisfied parity checks for that VN. Only the nodes with the maximum value of the objective function are "suspected" to be in error, and are flipped in that iteration. Although this approach results in faster convergence and increased correction capability when compared with typical bit-flipping-based algorithms [15,16], there are error patterns for which the decoding halts because a local minimum of the GDBF objective function is reached. One way to alleviate this drawback is to introduce randomness in VN or CN update functions. The algorithm that improves the decoder performance over the binary symmetric channel (BSC) by adding random binary sequences to the bit decisions during iterations is known as the probabilistic gradient descent bit-flipping (PGDBF) algorithm [17]. The impacts of randomness on performance improvement in the additive white Gaussian noise (AWGN) channel is analyzed in [18]. Another effect that adversely impacts convergence is the oscillatory behavior of a decoder—ns in a periodic fashion, and the algorithm loops endlessly. To avoid the "loops" in GDBF decoding, various approaches are proposed in the literature. An escaping scheme based on syndrome weight is proposed in [19]. In our work [20], we show that the same suspicious positions in subsequent iterations indicate the uncorrectable error patterns for the GDBF algorithm, known as *trapping sets*. In the same paper, we explained that the negative effect of the trapping sets can be avoided by applying multiple decoding attempts

and random re-initialization (MUDRI). In fact, if the decoder is trapped in a local minima of the objective function, random perturbations can help the decoder to escape from these minima, and converge to a correct codeword. If the randomness is added to all VNs, the PGDBF decoder with restarts and re-initialization can approach the maximum likelihood (ML) performance after a sufficiently large number of iterations [21]. The multiple decoding with random re-initializations can also be applied to the other hard-decision decoders, and the corresponding theoretical analysis is given for the Gallager-B decoder in [22,23].

Speeding up the PGDBF decoder convergence has been studied by numerous research groups in recent years. One approach is to use the threshold that corresponds to the maximum value of the objective function from the previous iteration, as proposed in [24]. Another method is to modify the objective function to include the information about the number of flipping for every VN as in [25]. In the taboo-list random GDBF (TRGDBF) [26,27], a binary random sequence is added to the objective function, and VNs flipped in the previous iteration are prevented from flipping in the current iteration. If every VN has knowledge about the number of iterations since the last flip, this knowledge can also be used in the objective function to further improve the performance [28]. All these results indicate that a time dimension of every particular VN plays a significant role in the decision process, and this can be further optimized similarly as in the concept of finite alphabet iterative decoders (FAID) [11]. This observation is at the heart of our method too.

However, the algorithms that apply randomness have increased complexity, as pseudo random number generators, producing independent "random" sequences in every iteration, are required for every VN. It has been shown that hardware implementation of these algorithms is not a trivial task [27,29,30]; however, as shown in [31], deterministic improvements of the GDBF—typically based on adaptive thresholds—are effective for high-variable-degree codes only. The goal of this paper is to design a deterministic algorithm applicable to LDPC codes with low variable degree (less than five) with low implementation complexity and high convergence speed.

1.2. Summary and Organization

In this paper, we propose an algorithm that utilizes the reliability information passed from neighboring VNs. The name Suspicion Distillation Gradient Descent Bit-Flipping Algorithm (SDGDBF) reflects its main feature that in every iteration to every VN we assign a level of suspicion about its current bit value. How much we suspect a VN is not only used to decide whether a bit of this VN will be flipped, but also in each iteration, we determine the number of satisfied and unsatisfied checks that connect that VN with the other suspected VNs. In this way, in the course of iterations, we "distill" such suspicious bits and flip them. We show that information if the VN was flipped in the previous iterations can be combined with this variable-suspicion information to provide a significant performance improvement for various LDPC codes. Moreover, the algorithm is completely deterministic, i.e., it does not require randomness in any step during the decoding process.

The rest of the paper is organized as follows. In Section 2, the generalized GDBF algorithm for the BSC is presented, and an overview of the trapping sets is given. The SDGDBF algorithm is proposed in Section 3, where a few introductory examples are given, modification of the flipping rule is proposed, and the algorithm is formally defined. The complexity analysis is given in Section 4. Numerical results obtained by Monte Carlo simulations are presented in Section 5 for various LDPC codes. Optimization of the decoder parameters and further work is given in Section 6. The final conclusions are outlined in Section 7.

2. GDBF Decoding and the Impact of Trapping Sets

At the encoder side, a (n, k) binary LDPC code $\mathcal{C}$ of rate $R = k/n$ is applied, where the information word with length k is represented with the corresponding codeword $\mathbf{x}$ with length n. The corresponding Tanner graph G consists of the set of variable nodes $V = \{v_1, v_2, \ldots, v_n\}$ and the set of check nodes $C = \{c_1, c_2, \ldots, c_m\}$. Parity check matrix H

is the bi-adjacency matrix of G. Two nodes in G are neighbors if there is an edge between them, and the degree of a node is the number of its neighbors. In this paper, we consider regular LDPC codes where each variable node has degree γ and every check node has degree ρ. The sets of neighbors of nodes v_i and c_j are denoted as $\mathcal{N}_{v_i}$ and $\mathcal{N}_{c_j}$, respectively.

The transmission of a codeword $\mathbf{x}$ through the BSC with crossover probability α_{BSC} is modeled as a modulo-2 addition of a Bernoulli $B(1;\alpha_{BSC})$ random sequence of errors $\mathbf{e}$, $\Pr(e_i = 1) = \alpha_{BSC}$. The received word at the output of the BSC channel is then $\mathbf{y} = \mathbf{x} \oplus \mathbf{e}$, where $\oplus$ denotes logical XOR operation, i.e., bitwise addition modulo 2.

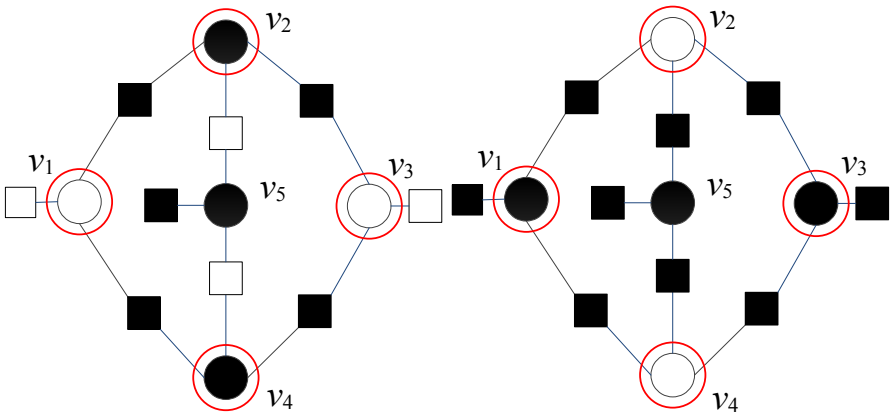
In the rest of this section, the decoder input will be denoted by $\mathbf{r}$. At the start of the decoding, the input to the decoder is equal to the channel output ($\mathbf{r} = \mathbf{y}$). In a more general framework, where the overall decoding process can be divided into multiple stages, we allow the decoder input to be re-initialized to another value during the decoding (it will be further explained in the next section); therefore, the initial estimation $\hat{\mathbf{x}}^{(0)}$ will be equal to a reference $\mathbf{r}$ in the analysis given below.

For the decoder input $\mathbf{r}$, the decoder calculates the energy function for the i-th variable node in the ℓ-th iteration [14,17] as

$$\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r}) = \hat{x}_i^{(\ell)} \oplus r_i + \sum_{c_j \in \mathcal{N}_{v_i}} \bigoplus_{v_k \in \mathcal{N}_{c_j}} \hat{x}_k^{(\ell)}, \tag{1}$$

where $\bigoplus$ denotes modulo-2 summation. Only those bits that have the maximum value of $\Lambda_i^{(\ell)}$ are called *suspicious nodes*, which means that they are most likely erroneous. Suspicious VNs are flipped in every iteration. The decoding is finished if all parity checks are satisfied or if the maximum number of decoding iterations, denoted by L, is elapsed.

Even if the number of erroneous variables is low, they might be positioned in such a way that causes decoding oscillations and prevents convergence of the decoder to a valid codeword. This situation is illustrated in Figure 1, where correct variable nodes are represented with white circles ○, while erroneous variable nodes are represented with black circles ●. Satisfied parity checks are represented by white squares □, while unsatisfied parity checks are represented by using black squares ■. Variable nodes v_1, v_2, v_3, and v_4 are suspicious both in the first and second iterations (with $\Lambda_i^{(1)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r}) = 2$ and $\Lambda_i^{(2)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r}) = 4$, for $i = 1, 2, 3, 4$); therefore, the corresponding three-bit error pattern cannot be corrected.



(**a**) $l = 1$ (**b**) $l = 2$

Figure 1. Decoding of three-bit error pattern by using GDBF decoder; flipping decisions after the first and the second iteration.

In the PGDBF algorithm [17], the suspicious VNs are flipped with a predefined probability p. It is necessary to generate Bernoulli $B(1;p)$ random variables $a_i^{(\ell)}$ ($\forall i, \ell$); only the variable nodes with $a_i^{(\ell)} = 1$ and maximum value of $\Lambda_i^{(1)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r})$ are flipped. In the MUDRI

algorithm, the decoding is performed in multiple stages, where a randomized received word is used as the decoder input in every particular decoding attempt.

Further improvement is proposed in the TRGDBF algorithm [27], where Bernoulli $B(1;p)$ random variable λ_i is added to the objective function, and taboo list T_i of the VNs flipped in the previous iteration is created. If $T_i = 1$ the corresponding node is prevented from flipping, and this can be obtained if we set $\Lambda_i(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r})^{(\ell)} = 0$ for variable nodes on the tabu list. For the TRGDBF, the energy function can be written in the following form:

$$\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r}) = \left(\hat{x}_i^{(\ell)} \oplus r_i + \sum_{c_j \in \mathcal{N}_{v_i}} \bigoplus_{v_k \in \mathcal{N}_{c_j}} \hat{x}_k^{(\ell)} + \lambda_i \right) (1 - T_i). \tag{2}$$

In a recent paper [28], the GDBF algorithm with momentum (GDBF-w/m) is proposed. The number of iterations from the previous flipping of v_i is denoted by w_i, and the corresponding momentum vector is defined as $\mu = (\mu(1), \mu(2), \ldots, \mu(w_{\max}))$, where we assume $\mu(w_i) = 0$ for $w_i > w_{\max}$. It has been shown in [28] that the convergence speed can be improved by using the correlation coefficient α. With the aim to easily provide integer values of energy functions, we introduce another coefficient β, as in the following expression

$$\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r}) = \alpha(\hat{x}_i^{(\ell)} \oplus r_i) + \beta \left(\sum_{c_j \in \mathcal{N}_{v_i}} \bigoplus_{v_k \in \mathcal{N}_{c_j}} \hat{x}_k^{(\ell)} \right) - \mu(w_i). \tag{3}$$

All of the above improvements of the GDBF algorithm can be summarized in Algorithm 1. The energy function is calculated according to the Equation (3). We set $\mathbf{r} = \mathbf{y}$ in all algorithms except the MUDRI, where the decoder input $\mathbf{r}$ is re-initialized at every decoding attempt. For the GDBF, PGDBF, and MUDRI we set $\alpha = 1, \beta = 1, \mu = \mathbf{0}$, and for the TRGDBF we set $\alpha = 1, \beta = 1, \mu = \gamma + 1$. In the PGDBF, MUDRI, and TRGDBF algorithms, we set $p < 1$ to incorporate randomness in the decoding process.

Algorithm 1 Generalized GDBF Algorithm

Input: $\mathbf{r}, p, \alpha, \beta, \mu$
$\forall v_i \in V: \ \hat{x}_i^{(0)} \leftarrow r_i$
$\mathbf{s}^{(0)} \leftarrow \hat{\mathbf{x}}^{(0)} H^T \left(\forall c_j \in C: \ s_c^{(0)} \leftarrow \bigoplus_{j \in \mathcal{N}_j} \hat{x}_j^{(0)} \right)$
$\ell = 0$
while $\mathbf{s}^{(\ell)} \neq \mathbf{0}$ and $l \leq L$ **do**
  $\forall i \in V$: Compute $\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r}))$ for given α, β, μ
  $\Lambda_{\max}^{(\ell)} \leftarrow \max_i(\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r})))$
  for $i \leftarrow 1$ to n **do**
    $a_i^{(\ell)} = B(1;p)$
    if $\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \mathbf{r}) = \Lambda_{\max}^{(\ell)}$ **then**
      $\hat{x}_i^{(\ell+1)} \leftarrow (1 \otimes a_i^{(\ell)}) \oplus \hat{x}_i^{(\ell)}$
    else
      $\hat{x}_i^{(\ell+1)} \leftarrow \hat{x}_i^{(\ell)}$
    end if
  end for
  $\mathbf{s}^{(\ell+1)} \leftarrow \hat{\mathbf{x}}^{(\ell+1)} H^T$
  $l \leftarrow l + 1$
end while
Output: $\hat{\mathbf{x}}^{(\ell)}$

3. Suspicion Distillation GDBF Algorithm

Although the algorithms that introduce a randomness into the decoding process can be very effective, generating independent random sequences in all iterations increase complexity and there is a need for a deterministic algorithm with similar performances.

The suspicion distillation GDBF (SDGDBF) algorithm uses the existing GDBF algorithms as its components, and in some iterations invokes a procedure for locating satisfied parity checks connected to erroneous variable nodes and unsatisfied checks connected to correct variable nodes. This procedure, as well as the rule in which (a small fraction of) iterations are used to invoke it, is the main novelty in the paper. In those iterations when a modification is applied, we set $\mu = \mathbf{0}$. The intuition behind this is that the graph structure plays a more important role in these iterations than the messages in the previous iterations. In such an iteration, the reference is equal to the input estimate $\hat{\mathbf{x}}^{(\ell)}$, and therefore $\hat{\mathbf{x}}^{(\ell)} \oplus \mathbf{r} = 0$. It is clear that the energy function is equal to the number of unsatisfied checks if $\beta = 1$ (for any value of α). The output of the modification will be the updated reference $\mathbf{r}$.

3.1. Modification of the Flipping Rule

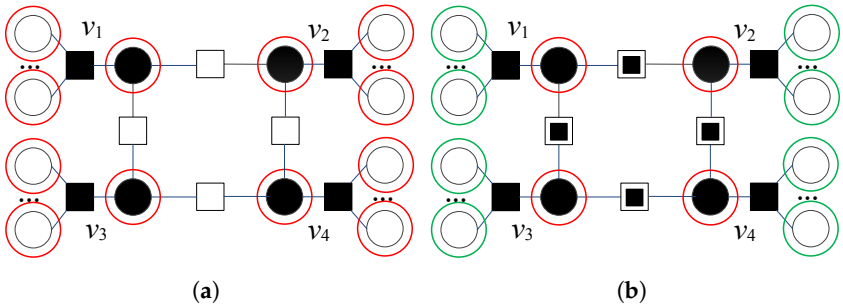
The proposed modification follows the basic steps:

1. For the given codeword estimate $\hat{\mathbf{x}}^{(\ell)}$, the set of candidates for flipping is expanded. The set of *suspicious* variable nodes $V_S^{(\ell)}$ contains VNs with $\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) \geq \Lambda_{\max 2}^{(\ell)}$. $\Lambda_{\max 2}^{(\ell)} = \underset{i}{\text{2ndmax}}(\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}))$ denotes the second maximum value of the energy function, wherein the function 2ndmax(.) computes the second maximum value of its vector-valued argument. We set the indicator that the VN is suspicious $I_{S,i}^{(\ell)} = 1$ if $v_i \in V_S$ (and equal to zero otherwise). The nodes with the energy function equal to $\Lambda_{\max}^{(\ell)}$ form a set of *very suspicious* variable nodes, denoted by $V_{VS}^{(\ell)}$;
2. For all VNs, we identify neighboring parity checks that are satisfied ($c_j = 0$) and have at least one more suspicious VNs as their own neighbor. The number of such *unreliable satisfied* checks for the i-th variable node in the ℓ-th iteration is denoted as $N_{SC,i}^{(\ell)}$. The energy function is set to $\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) = \gamma$ if the condition $N_{SC,i}^{(\ell)} + \Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) = \gamma$ is satisfied. In such a case, we add v_i to the set $V_S^{(\ell)}$;
3. For $v_i \in V_S^{(\ell)}$, we update $N_{SC,i}^{(\ell)}$ and recalculate $\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) \leftarrow \Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) + N_{SC,i}^{(\ell)}$. We update $\Lambda_{\max}^{(\ell)}$ and the corresponding set $V_{VS}^{(\ell)}$;
4. For $v_i \in V_{VS}^{(\ell)}$, we identify parity check neighbors that are not satisfied ($c_j = 1$), and have at least one more very suspicious VN as their neighbor. The number of such *unreliable unsatisfied* checks for the i-th variable node in the ℓ-th iteration is denoted as $N_{NC,i}^{(\ell)}$, and number of all unsatisfied checks is denoted as $N_{UC,i}^{(\ell)}$. If $N_{NC,i}^{(\ell)} = N_{UC,i}^{(\ell)}$, we recalculate $\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) \leftarrow \Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) - N_{NC,i}^{(\ell)}$, and update $V_{VS}^{(\ell)}$;
5. The node v_i is flipped if $v_i \in V_{VS}$.

The modification is used in certain iterations to improve the reliability of the flipping decisions by updating the reference $\mathbf{r}$, and it can be applied to any GDBF-based algorithm. In this section, we show a few illustrative examples of when the modification is applied to the standard GDBF algorithm. It can be noticed that, by applying the proposed modification, the decoder now searches for the parity checks that are misleading for the standard bit-flipping algorithm, i.e., for the parity checks that we call unreliable:

- Unreliable satisfied parity check, denoted by ▣, is satisfied but does not represent a reliable indicator that v_i is correct;
- Unreliable unsatisfied parity check, denoted by ◘, is unsatisfied but does not represent a reliable indicator that v_i is erroneous;
- Reliable parity checks are those that do not belong to the above two unreliable categories, and are denoted in a typical fashion—□ is a reliable satisfied check, while ■ is a reliable unsatisfied check.

In Figure 2a, a typical four-bit error pattern for a code with girth $g = 8$ and variable node degree $\gamma = 3$ is presented. Variable nodes v_1, v_2, v_3, and v_4, as well as the other variable nodes connected to unsatisfied checks, are very suspicious both in the first and the second iteration (as $\Lambda^{(1)}_{\max 2} = 0$, we can notice that $V^{(\ell)}_{VS} = V^{(\ell)}_{S}$). Very suspicious VNs are marked with red circles. This four-bit error pattern is uncorrectable using the GDBF, as the GDBF decoder in all further iterations flips the same positions. We identify unreliable satisfied parity checks for every particular VN. Two unreliable satisfied checks (denoted by ▣) are neighbors of node v_1—one of them is connected to the suspicious node v_2, and the other is connected to the suspicious node v_3. A similar situation is found with the nodes v_2, v_3, and v_4; therefore, $N^{(1)}_{SC,i} = 2$ and after the recalculation, we obtain $\Lambda^{(1)}_i(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}) = 3$, for $i = 1, 2, 3, 4$ (for the other VNs, the energy function is not updated). VNs that correspond to the updated set V_{VS} are shown in Figure 2b, marked with red circles, while green circles correspond to the other VNs that belong to the updated set V_S. This four-error pattern is corrected in the same iteration when the modification is applied.

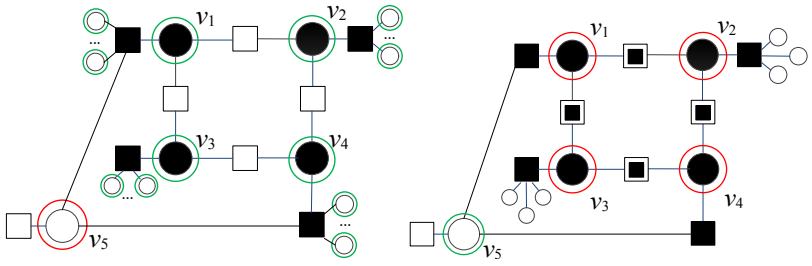


(**a**) (**b**)

Figure 2. Decoding of four-bit error pattern with four very suspicious variable nodes, flipping decisions after the first iteration of GDBF, without modification and after the applied modification. Red circles correspond to very suspicious nodes; green circles to suspicious nodes. (**a**) without modification. (**b**) applied modification.

The graph that corresponds to another four-bit error pattern is given in Figure 3a. Node v_5 has two unsatisfied parity checks, and only this node is very suspicious. In the next iteration, the energy function for v_5 is equal to $\Lambda^{(2)}_5(\hat{\mathbf{x}}^{(2)}, \mathbf{r}) = 2$. As the same VN is suspicious in both iterations, this pattern cannot be corrected without a modification.

In this example, the only very suspicious node is v_5 (marked with a red circle); the other suspicious nodes are all VNs connected with one unsatisfied check (marked with green circles). There are $N^{(1)}_{SC,i} = 2$ unreliable satisfied parity checks that connect v_1 with suspicious nodes v_2 and v_3. These satisfied checks are no more assumed as valid indicators if the node v_1 is correct, and the corresponding energy function is recalculated as $\Lambda^{(1)}_1(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}) = 3$. Repeating the same procedure for the other variable nodes, we easily obtain that the very suspicious nodes are now v_1, v_2, v_3, and v_4 (Figure 3b).



(**a**) (**b**)

Figure 3. Decoding of four-bit error pattern with one very suspicious variable node and a large number of less suspicious-looking variable nodes, flipping decisions after the first iteration of the GDBF, without modification and after applying the modification. (**a**) Without modification. (**b**) Applied modification.

In the following example, we analyze the correction of the seven-bit error pattern and illustrate a situation in which a node outside $V_S^{(l)}$ is a flipping candidate (Figure 4a).

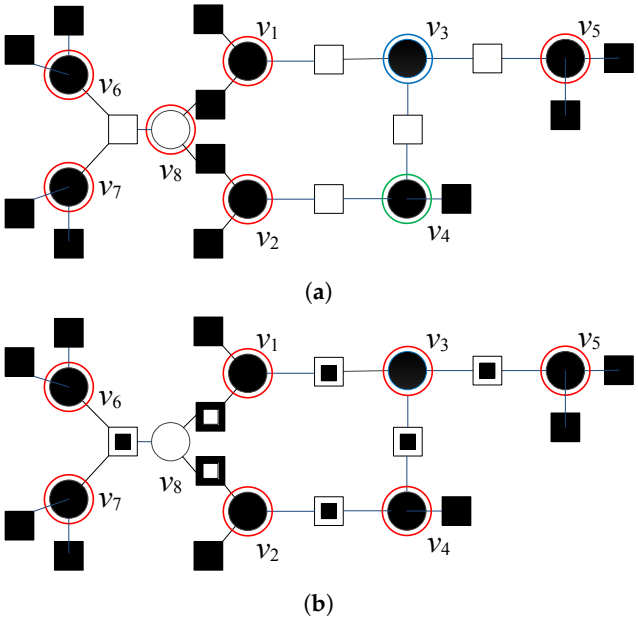


Figure 4. Decoding of seven-bit error pattern with four very suspicious variable nodes, after the first iteration of GDBF, without modification and after the applied modification. The blue circle corresponds to the node that should be added in the sets V_S and V_{VS}. (**a**) Without modification. (**b**) Applied modification.

Red circles denote six very suspicious nodes, with an energy function equal to $\Lambda_{\max}^{(1)} = 2$, and green circle denotes another suspicious node v_4 with $\Lambda_{\max 2}^{(1)} = 1$. If we concentrate on v_3, we will see that it has all three satisfied parity checks as its neighbors. Each of these checks are connected to one suspicious VN. According to the proposed modification, these parity checks cannot be considered valid indicators of the correctness of v_3. These checks are satisfied but unreliable, and we obtain $N_{SC,3}^{(1)} = 3$. Although v_3 is initially not a suspicious node ($\Lambda_3^{(1)}(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}) = 0$), it satisfies the condition $\Lambda_3^{(1)}(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}) + N_{SC,3}^{(1)} = \gamma$, meaning that among the neighbors of the node v_3 there are not any reliable satisfied checks. Thus, we mark v_3 by a blue circle. Following the modification rule presented in step 2, we recalculate $\Lambda_3^{(1)}(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}) = 3$, and this node will be added in sets $V_S^{(\ell)}$ and $V_{VS}^{(\ell)}$.

According to step 3, the energy function values for the other suspicious nodes are also updated ($\Lambda_i^{(1)}(\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(1)}) = 3$, $i = 1, 2, 4, 5, 6, 7, 8$). Notice that the parity check that connects v_2 and v_4 is also unreliable satisfied. The set $V_{VS}^{(\ell)}$ is updated.

Finally, node v_8 is connected with $N_{UC,8}^{(1)} = 2$ unsatisfied checks. These parity checks are unreliable unsatisfied (denoted by ◘ in Figure 4b), which have at least one more neighbor that is very suspicious, i.e., $N_{NC,8}^{(1)} = 2$. As all unsatisfied checks are unreliable, we update $\Lambda_8^{(1)}(\hat{\mathbf{x}}, \mathbf{r}) = 1$. This node is removed from V_{VS}, and all errors in the pattern are corrected (see red circles in Figure 4b).

The exact formulation of the flipping rule modification is presented in Algorithm 2. The sets V_{VS} and V_S are initially created according to the value of the energy function, for $\mathbf{r} = \mathbf{x}^{(\ell)}$. Both sets are extended with the variable nodes v_i that satisfy the condition defined in step 2, and the energy function of the suspicious nodes are updated following the rule from step 3. After the recalculation of the energy functions, we remove v_i from V_{VS} if it satisfies the condition from step 4. As explained later, the proposed modification is applied only in certain iterations.

Algorithm 2 Modification of the flipping rule

1: **Input:** $\hat{\mathbf{x}}^{(\ell)}$
2: $\forall v_i \in V$: Compute $\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})$ for $\alpha = 0, \beta = 1$ and $\rho = 0$
3: $\Lambda_{\max}^{(\ell)} \leftarrow \max_i(\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})), V_{VS}^{(\ell)} = \{v_i | \Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) = \Lambda_{\max}^{(\ell)}\}$
4: $\Lambda_{\max 2}^{(\ell)} \leftarrow 2\text{nd}\max_i(\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})), V_S^{(\ell)} = \{v_i | \Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) \geq \Lambda_{\max 2}^{(\ell)}\}, I_{S,i}^{(\ell)} = 0 \; \forall i$
5: **for** $i \leftarrow 1$ to n **do**
6: **if** $v_i \in V_S^{(\ell)}$ **then**
7: $I_{S,i}^{(\ell)} = 1$
8: **end if**
9: **end for**
10: **for** $i \leftarrow 1$ to n **do**
11: $N_{SC,i}^{(\ell)} = \sum_{c_j=0 \wedge c_j \in \mathcal{N}_{v_i}} \vee_{v_k \in \mathcal{N}_{c_j}/v_i} I_{S,k}^{(\ell)}$
12: **if** $N_{SC,i}^{(\ell)} + \Lambda_i^{(\ell)}((\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})) = \gamma$ **then**
13: $\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) = \gamma, V_{VS}^{(\ell)} = V_{VS}^{(\ell)} \cup v_i, V_S^{(\ell)} = V_S^{(\ell)} \cup v_i, I_{S,i}^{(\ell)} = 1$
14: **end if**
15: **end for**
16: **for** $i \leftarrow 1$ to n **do**
17: **if** $v_i \in V_S^{(\ell)}$ **then**
18: $N_{SC,i}^{(\ell)} = \sum_{c_j=0 \wedge c_j \in \mathcal{N}_{v_i}} \vee_{v_k \in \mathcal{N}_{c_j}/v_i} I_{S,k}^{(\ell)}$
19: $\Lambda_i^{(\ell)}((\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})) \leftarrow \Lambda_i^{(\ell)}((\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})) + N_{SC,i}^{(\ell)}$
20: **end if**
21: **end for**
22: $\Lambda_{\max}^{(\ell)} \leftarrow \max_i(\Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})), V_{VS}^{(\ell)} = \{v_i | \Lambda_i^{(\ell)}(\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)}) = \Lambda_{\max}^{(\ell)}\}, I_{S,i}^{(\ell)} = 0 \; \forall i,$
23: **for** $i \leftarrow 1$ to n **do**
24: **if** $v_i \in V_{VS}^{(\ell)}$ **then**
25: $I_{S,i}^{(\ell)} = 1$
26: **end if**
27: **end for**
28: **for** $i \leftarrow 1$ to n **do**
29: **if** $v_i \in V_{VS}^{(\ell)}$ **then**
30: $N_{NC,i}^{(\ell)} = \sum_{c_j=1 \wedge c_j \in \mathcal{N}_{v_i}} \vee_{v_k \in \mathcal{N}_{c_j}/v_i} I_{S,k}^{(\ell)}$
31: $N_{UC,i}^{(\ell)} = \sum_{c_j \in \mathcal{N}_{v_i}} \oplus_{v_k \in \mathcal{N}_{c_j}} \hat{x}_k^{(\ell)}$
32: **if** $(N_{NC,i}^{(\ell)} > 0) \wedge (N_{NC,i}^{(\ell)} = N_{UC,i}^{(\ell)})$ **then**
33: $\Lambda_i^{(\ell)}((\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})) \leftarrow \Lambda_i^{(\ell)}((\hat{\mathbf{x}}^{(\ell)}, \hat{\mathbf{x}}^{(\ell)})) - N_{NC,i}^{(\ell)}$
34: **else**
35: $\hat{x}_i^{(\ell)} = \hat{x}_i \oplus 1$
36: **end if**
37: **end if**
38: **end for**
39: **Output:** $\mathbf{r} = \hat{\mathbf{x}}^{(\ell)}$

Lines 1–9 in Algorithm 2 correspond to the first step defined on page 6. Lines 10–15 correspond to the second step, where we extend the set of suspicious VNs with the VNs that have no reliable satisfied parity checks as their neighbors. Lines 16–27 correspond to the third step, where we update the energy function for suspicious VNs, as well as the set V_{VS}. Lines 28–38 correspond to the fourth and the fifth step defined on page 6. The value of the energy function for very suspicious VNs is reduced if it has no reliable unsatisfied parity checks among its neighbors. If this condition is not satisfied, we flip the very suspicious variable node. Line 39 defines the output of the algorithm.

3.2. Modification Strategy and the Reference Update

In this paper, we consider the design of the deterministic low-complexity GDBF-based algorithm with superior performance. To avoid an increase in complexity, it is desirable not to apply the previously proposed modifications in every iteration of the decoding process. In fact, the best performance is obtained if the modification is applied in the following cases: (a) only for the patterns that cannot be decoded by using the basic algorithm and (b) in the iterations when it is detected that the decoding algorithm does not converge.

If the GDBF based on Equation (1) is used as a basic algorithm, and the same positions are selected for flipping in two subsequent iterations (example from Figure 1), it is obvious that the decoding will be terminated without any success if the modification is not applied. To implement the corresponding "alarm", it is necessary to remember the flipping positions in at least two subsequent iterations.

If we use the GDBF-w/m as a basic algorithm, the trapping sets with short cycles will be corrected, and there is no need to check if all flipping positions are repeated after a certain number of iterations. After a certain number of iterations, even for this type of decoder, the probability of correcting any error pattern is negligible (this is illustrated in Section 5). This number of iterations after which no more bits are corrected is estimated empirically and denoted by K_1. K_1 determines the right moment to apply the modification described in Section 3.1.

The codeword estimate obtained after the applied modification can be used as a new reference $\mathbf{r}$; therefore, the modification formulated in Algorithm 2 can be considered as the function $\Omega(.)$ that converts a codeword estimate $\hat{\mathbf{x}}^{(\ell)}$ into the new reference $\mathbf{r}$, that will be used as the input of the basic decoder (with any GDBF-based algorithm) in the next decoding round, during the next K_2 iterations. Various decoding strategies are possible. Modification can be applied to the codeword estimate obtained after the first attempt of K_1 iterations of the GDBF-based algorithm, denoted by $\hat{\mathbf{x}}_I^{(K_1)}$. In this scenario, the modification should be applied after every decoding attempt, as illustrated in Figure 5a; however, this is not useful if the Hamming distance between the codeword and its estimate is too large. The other approach would be to apply successive modifications to the received word, and use the corresponding vectors as the references in multiple decoding attempts (Figure 5b). We focus on this approach in the next section.

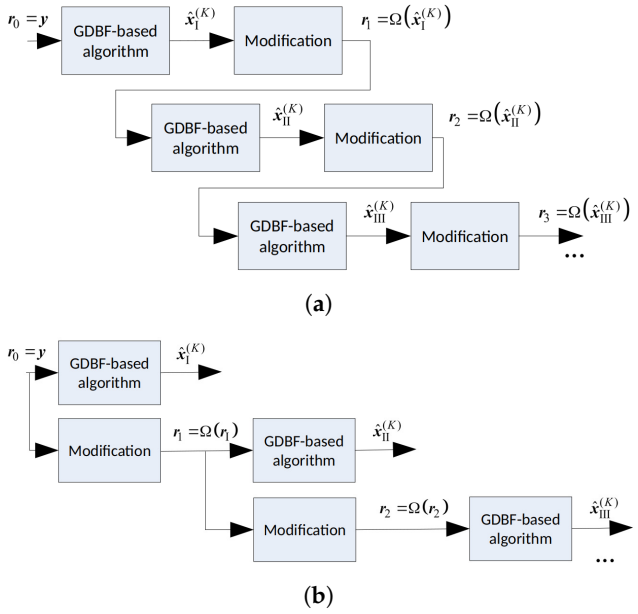


Figure 5. Reference updating after the decoding rounds (K iterations each); two approaches. (**a**) Modifications applied on the estimates. (**b**) Successive modifications applied on the received word.

3.3. The Decoder Input Re-Initializations

In our previous work, we noticed that the random re-initializations of the decoder input combined with the multiple decoding attempts resulted in more significant performance improvement [20].

In this paper, we apply a deterministic re-initialization, where the reference update can be performed by using a deterministic function $\Phi_q(.)$ that transforms the received word $\mathbf{y}$ into the reference $\mathbf{r}_q$. This reference will further be used as the initial estimate in the corresponding decoding attempt, as illustrated in Figure 6. The decoding trajectories are given in Figure 7. The re-initializations are applied only if the basic GDBF decoder fails to decode the codeword after K_1 iterations (decoding trajectory represented with black lines). Red dots represent the updated references that should be used as the input of the basic GDBF decoder in the corresponding re-initialization, and arrows without additional notations correspond to one iteration of the basic GDBF decoder, described in Algorithm 1.

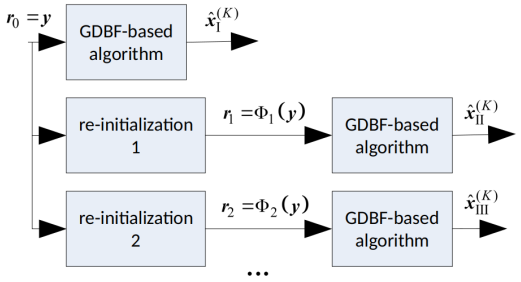


Figure 6. Reference updates by using the deterministic re-initializations; illustration of the multiple decoding attempts.

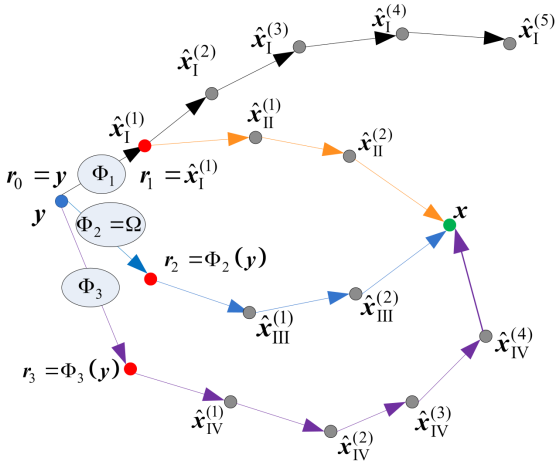


Figure 7. Various strategies for the decoder input re-initializations.

The following approaches, illustrated in Figure 7, have been shown to be very effective:

- The codeword estimate after the first iteration is used as a new input for the decoder, i.e., flipping decisions in the first iteration define how to obtain the reference from $\mathbf{y}$ (the decoding trajectory represented by an orange color in Figure 7), and the function $\Phi_1(.)$ that assigns $\mathbf{y}$ to reference $\mathbf{r}_1$ corresponds to one iteration of the GDBF decoding algorithm described in Algorithm 1;
- The modification described in Algorithm 2 can also be considered as a special case of the reference re-initialization if we apply transformation $\mathbf{r}_2 = \Phi_2(\mathbf{y}) = \Omega(\mathbf{y})$ (the decoding trajectory represented by the blue color in Figure 7);
- The reference $\mathbf{r}_3$ used in the purple trajectory represents the received sequence $\mathbf{y}$ changed in a single bit position. The position of the received bit that will be changed (flipped) is chosen among bits that were flipped during the first few iterations of the basic algorithm run prior to the modification.

Finally, the overall flowchart of the SDGDBF algorithm is presented in Figure 8. It is based on Algorithm 1, where the energy function is calculated by Equation (3). After K iterations, modification defined in Algorithm 2 is applied in one iteration, and the reference is updated. If the modification is repeated Z times, the re-initialization is applied. The decoding stops if a codeword is reached, or if a maximum number of iterations is elapsed. As we show in the next section, even if parameters K and Z are determined empirically, a significant performance improvement compared to the existing GDBF-based algorithms is possible.

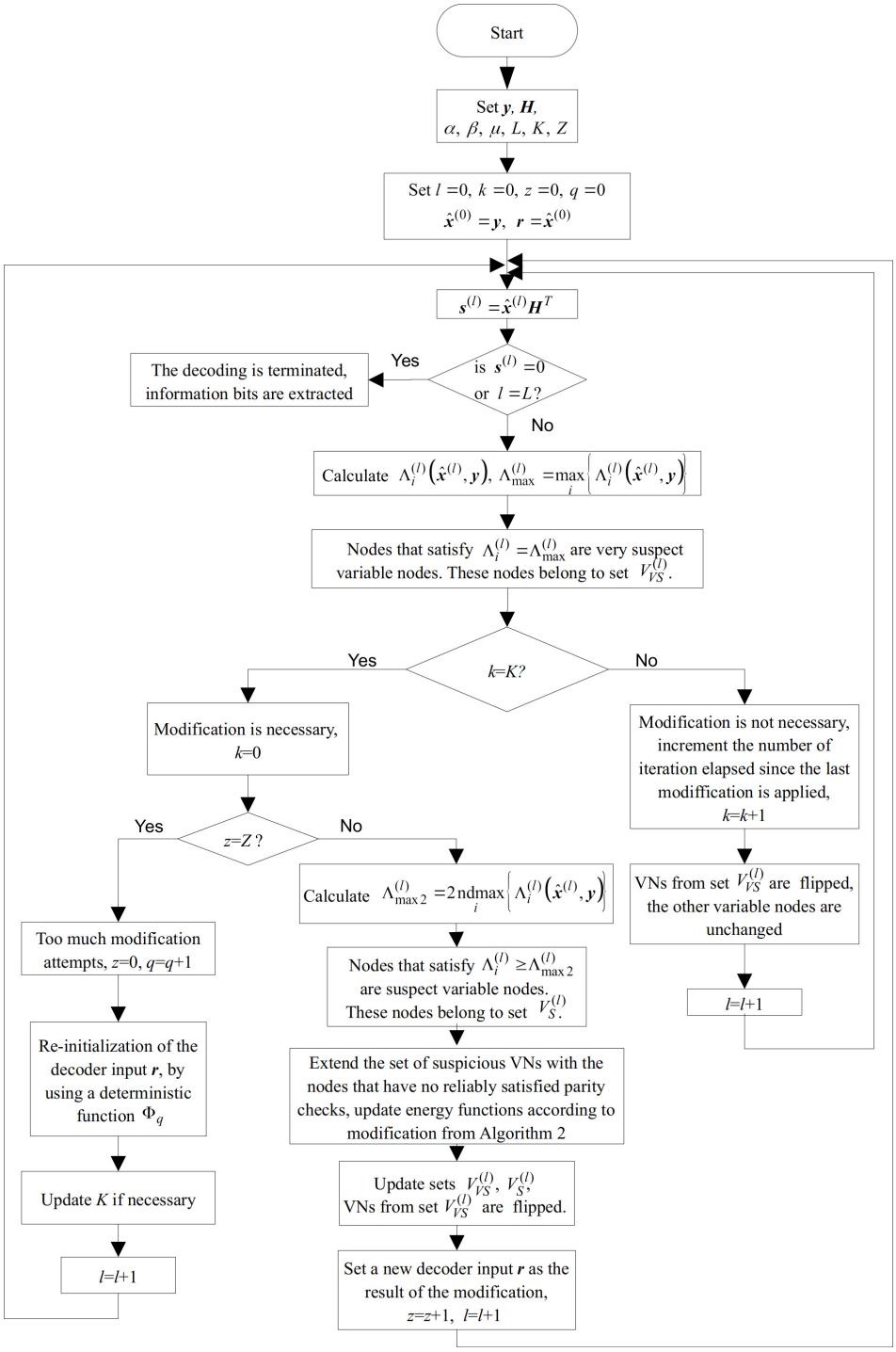


Figure 8. The flowchart of the SDGDBF algorithm.

4. Implementation Complexity

4.1. Computational Complexity

In this subsection, we examine the complexity of the SDGDBF decoder and make comparisons with the state-of-the-art GDBF and PGDBF decoders. Without going into a specific hardware realization, we estimate the number of arithmetical and logical operations required to perform a single decoding iteration. According to Algorithm 1, there are four types of operations in the GDBF-w/m decoder: binary XOR operations with ρ inputs, integer additions, a global maximization (with n inputs), and integer comparisons (Table 1). The global maximization is implemented by $n-1$ integer comparisons, and it was estimated that the GDBF has 40% larger complexity when compared to the BF algorithm [12]. The GDBF-w/m decoder has n extra integer additions (with the momentum vector).

Table 1. Computational complexity per decoding iteration.

Type of Operation	GDBF-w/m (Algorithm 1)	The Modification (Algorithm 2)
ρ-input XOR	m	m
$(\rho-1)$-input OR	-	$<3m$
1-bit comparison	-	$<2n$
Integer addition	$2n$	$<5n$
Global maximization	1	3
Integer comparison	n	$<5n$

The modification, proposed in Algorithm 2, contains all operations of the GDBF-w/m decoder; however, it also requires computational resources to perform: (i) integer additions, which are proportional to sizes of sets V_S and V_{VS} (both are lower than n); (ii) two additional global maximization operations as well as additional integer comparators; (iii) binary OR operations and single bit comparators. Given the fact that the complexity of the modification varies depending on the decoded sequence, in Table 1 we give an upper complexity bound. It should be noted that in the complexity analysis, we neglect possible integer multiplications, required for α and β scaling, given the fact that α and β are usually small integers (the most often values are $\alpha = \beta = 1$), and that such scaling can be implemented through decimal point shifting, or small lookup tables. We also do not include logical circuits for syndrome calculation, nor do we include circuits that count iterations from previous variables flips, needed for the momentum term calculations. All the neglected operations are common to the GDBF-w/m and the proposed modification, and do not significantly influence the relative complexity ratio between the two solutions.

Roughly, an iteration where the proposed modification is applied is three times more complex than one iteration of the GDBF-w/m decoder; however, we use the modification only in cases that the GDBF-w/m fails to decode the received sequence in K_1 iterations, meaning that Algorithm 2 is rarely used. For example, when the BSC crossover probability is equal to $\alpha_{\mathrm{BSC}} = 0.01$ and average number of iterations is equal to $I_{av} = 1.87$. In this case, the GDBF-w/m decoder applied on Tanner code for $K_1 = 25$ iterations lowers the FER to approximately 10^{-5}, meaning that the modification is employed, on average, only once in 10^5 decoded sequences. In the rest of the decoding process, the modification is periodically applied every $K_2 = 20$ iterations and the computational complexity at the worst case can be estimated as $C_{SDGDBF} = (19/20) \times (C_{GDBFwm}) + (1/20) \times (3C_{GDBFwm}) = 1.1 \times C_{GDBFwm}$, where C_{GDBFwm} denotes computational complexity of the GDBF-w/m decoder.

On the other hand, employing random generators in the PGDBF decoder through linear feedback shift registers, as the alternative to the approach proposed in this paper, leads to a nine times larger memory requirements and close to 60% computational increase, compared to the GDBF decoder [32]. The random generators are used in every iteration, which means that they influence evenly the worst case and average complexities. It should be emphasized that a less complex random generator architecture is proposed in [29]; however, it is uncertain how it will influence the performance of arbitrary chosen code.

4.2. Efficient Implementation

Figure 9 illustrates the basic implementation concept of the proposed algorithm. During the decoding, we use three n-bit memory registers:

- **RECEIVED WORD** contains the received word from the channel, and it is stored here for possible re-initializations.
- **REFERENCE** is used to store vector $\mathbf{r}$, which will be used as the decoder input in the current attempt. Initially, this register contains the received word from the channel ($\mathbf{r} = \mathbf{y}$), and its update is very simple. If re-initialization is applied, the reference is equal to a slightly changed received word, as explained in the previous section.
- **ESTIMATE** is the memory that contains the current estimation of the codeword, i.e., the estimation after the ℓ-th iteration, with initial value $\hat{\mathbf{x}}^{(\mathbf{0})} = \mathbf{y}$. In addition, the information if the node is suspicious or very suspicious is stored here.

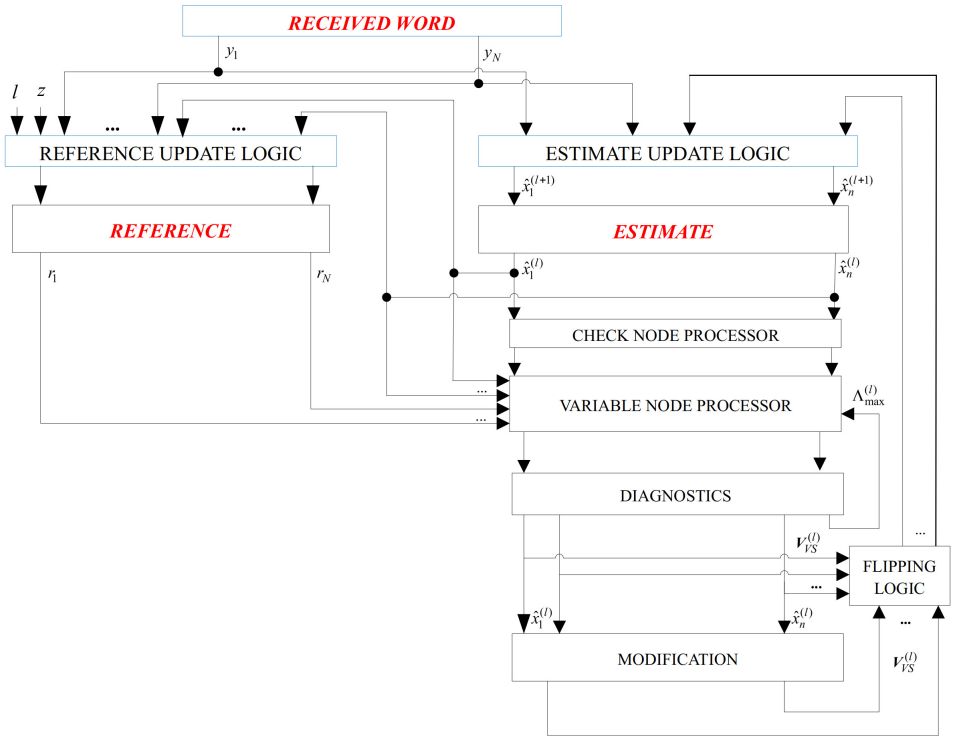


Figure 9. Main blocks in the implementation of the SDGDBF algorithm.

Parity checks for the estimated codeword are realized using XOR gates with ρ inputs. A variable node processor is realized using majority logic (MAJ) gates. **DIAGNOSTICS** uses counters to determine parameters l, k, z and logic gates that compare it with predefined values L, K, Z, respectively (to check if a modification is necessary). Further, it calculates the thresholds for MAJ gates in the ℓ-th iteration.

When compared to the GDBF-w/m algorithm, we have an additional n-bit memory register **REFERENCE** (in the GDBF, received word **y** is used as the reference in all iterations), memory for storing suspicious, and very suspicious positions. Block **MODIFICATION** is used for modifying the flipping rule (that is not used in all iterations). This block uses one OR logic gate with $\rho - 1$ inputs in each check node, as well as the integer adders that calculate $N_{SC,i}^{(\ell)}$ and $N_{NC,i}^{(\ell)}$; however, only m OR gates are required, as the $3m$ OR operations in steps 2–4 are consequently performed on the same logical gates. The same maximization circuit that is used to find the maximum of the energy functions in the GDBF-w/m algorithm can be also used to find the second maximum, and to update the maximum of the energy function in step 3. In addition, the same integer comparators and integer adders that are used in step 1 can be reused to perform integer comparisons in further steps.

5. Numerical Results

In this section, we illustrate the decoding performance of the SDGDBF decoder. It is compared with the decoding performance of the existing GDBF-based algorithms. In addition, the floating-point implementation performance of the sum-product algorithm (SPA) is included for reference. To make a fair comparison, the maximum number of iterations is set to $L = 300$ for all bit-flipping-based algorithms and $L = 50$ for the SPA algorithm, if not specified otherwise. The same assumption is used in the previous relevant papers [27,28], and it is based on the throughputs achieved in hardware implementations [33]. The numerical results are obtained by using Monte Carlo simulations, and the frame error rate (FER) estimation is terminated when at least 100 failed codewords are collected.

In all performed simulations, the GDBF algorithm with momentum is selected as a base for the SDGDBF decoder (the energy function is calculated by using expression Equation (3)). Although the modifications presented in Section 3 can also be applied to the other types of the decoders, we have selected the GDBF-w/m to reduce the decoder complexity as there is no need to apply a random generator and there is no need to detect the occurrence of uncorrectable error patterns with short cycles in this case.

We first compare the FER performance of the proposed algorithm with the other GDBF-based algorithms for the $(155, 64)$ Tanner code, with code rate $R = 0.4$. The code has a quasi-cyclic structure with construction proposed in papers [34,35] and it can be represented by using bipartite graph with $n = 155$ variable nodes and $m = 93$ check nodes, with VN degree $\gamma = 3$, CN degree $\rho = 5$, and girth $g = 8$.

The FER performance for the Tanner code is presented in Figure 10. For the GDBF-w/m and the SDGDBF, we set weighting factors $\alpha = 2$, $\beta = 2$, and momentum vector $\mu = [2, 1]$. In the SDGDBF algorithm, the modification described in Algorithm 2 is applied after $K = 25$ iterations. Further periodical re-initializations, based on the flipping of only one suspicious position in the received word (as described in Section 3.3), are applied on every $K = 10$ iterations and followed by $Z = 1$ modification each. This strategy resulted in superior performance in the error floor region and the SDGDBF with $L = 300$ iterations overcomes the performance of the floating-point SPA with $L = 50$ if $\alpha_{BSC} < 0.025$.

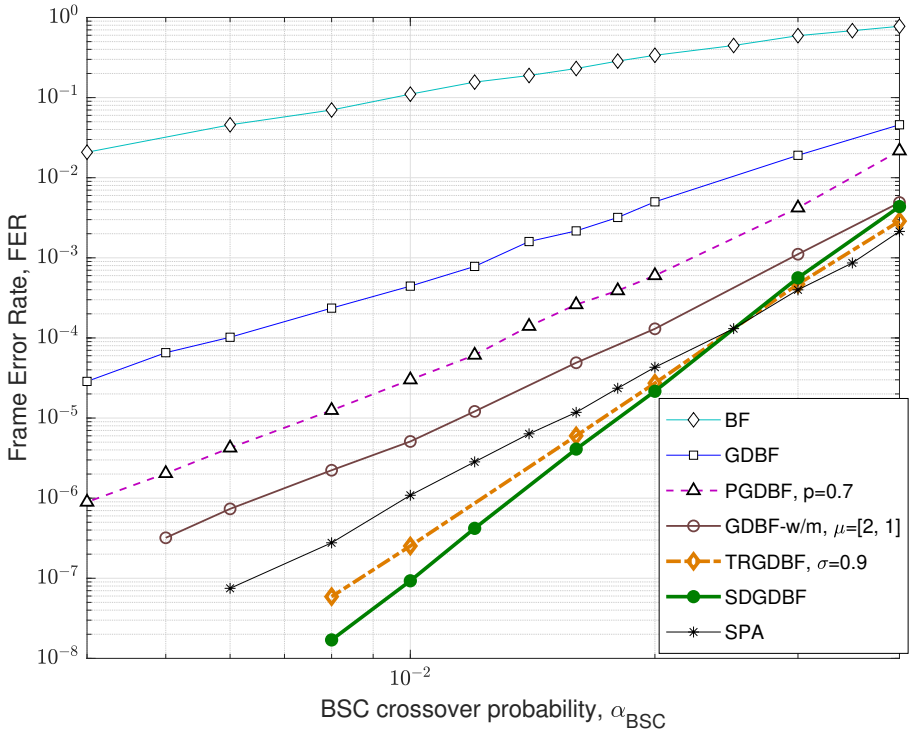


Figure 10. Performance of the various decoding algorithms, Tanner code ($n = 155$, $\gamma = 3$, $\rho = 5$).

The FER performance for the Tanner code for various values of L is presented in Figure 11, for crossover probability $\alpha_{BSC} = 0.01$. It is clearly visible that the deterministic hard-decision algorithms (GDBF, Gallager-B, and GDBF-w/m) do not improve performance after a certain number of iterations. The GDBF saturates after ten iterations, and further iterations cannot reduce the FER level anymore. The GDBF-w/m algorithm corrects most of the correctable error patterns in the first 20–30 iterations and saturates after 50 iterations; therefore, it is a wise strategy to apply the modification when the most of the correctable patterns are corrected by using the base algorithm. We empirically set the initial value $K_1 = 25$ in the case when the GDBF-w/m is chosen as a base algorithm. This value can be further reduced to $K_2 = 10$ after re-initializations are applied, as the benefit of frequent re-initialization is estimated to be higher than the benefit of a larger parameter K. It can be noticed that the SDGDBF algorithm has superior performance when compared with the MUDRI algorithm with random re-initializations on L_1 iterations, and the TRGDBF algorithm that combines random sequences with the tabu-list principle. In addition, it is visible that the SDGDBF algorithm with $L = 100$ has the same performance as the SPA algorithm with $L = 50$. If $L > 150$, the SDGDBF algorithm results in better performance than the SPA algorithm for the same value of L.

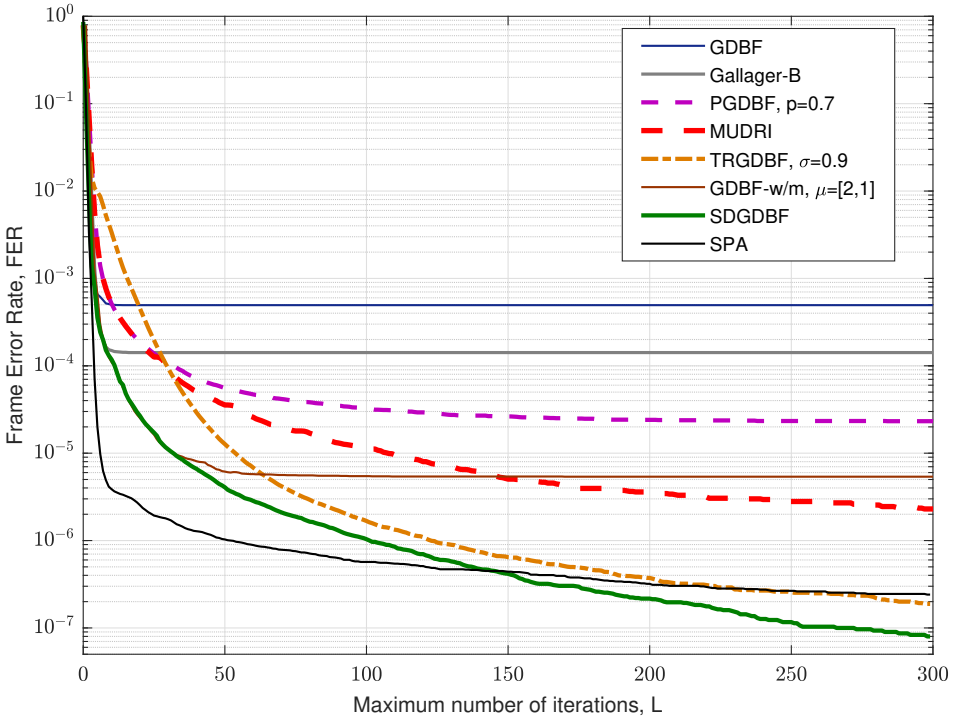


Figure 11. Convergence speed of the various decoding algorithms, Tanner code.

Performance of the analyzed GDBF-based algorithms for (1296, 648) QC-LDPC code with code rate $R = 0.5$ is presented in Figure 12. The corresponding construction method is presented in paper [33], and the code can be represented by using a bipartite graph with $n = 1296$ variable nodes and $m = 648$ parity checks, with VN degree $\gamma = 3$, CN degree, $\rho = 6$, and girth $g = 8$. For this code, it is obvious that the TRGDBF and the GDBF-w/m algorithms have much better performance when compared to the GDBF and the PGDBF algorithms. This illustrates the importance of the momentum for the longer codes. If the modification proposed in Algorithm 2 is applied to the GDBF-w/m, further performance improvement is possible if we choose the appropriate decoding parameters. The selection of the decoder parameters are explained with the help of the results presented in Figure 13 (given for $\alpha_{BSC} = 0.02$). The decoding convergence for the GDBF-w/m is much faster compared to the TRGDBF algorithm, and the GDBF-w/m provides lower FER values when compared to the GDBF and the PGDBF for any value of L.

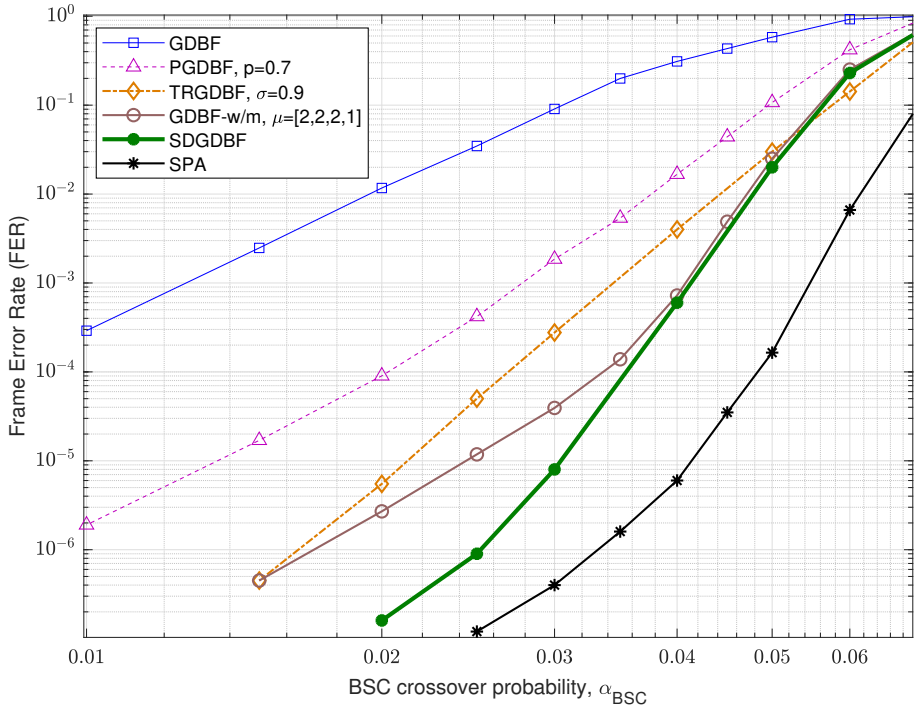


Figure 12. Performance of the various decoding algorithms, QC-LDPC code ($n = 1296$, $\gamma = 3$, $\rho = 6$).

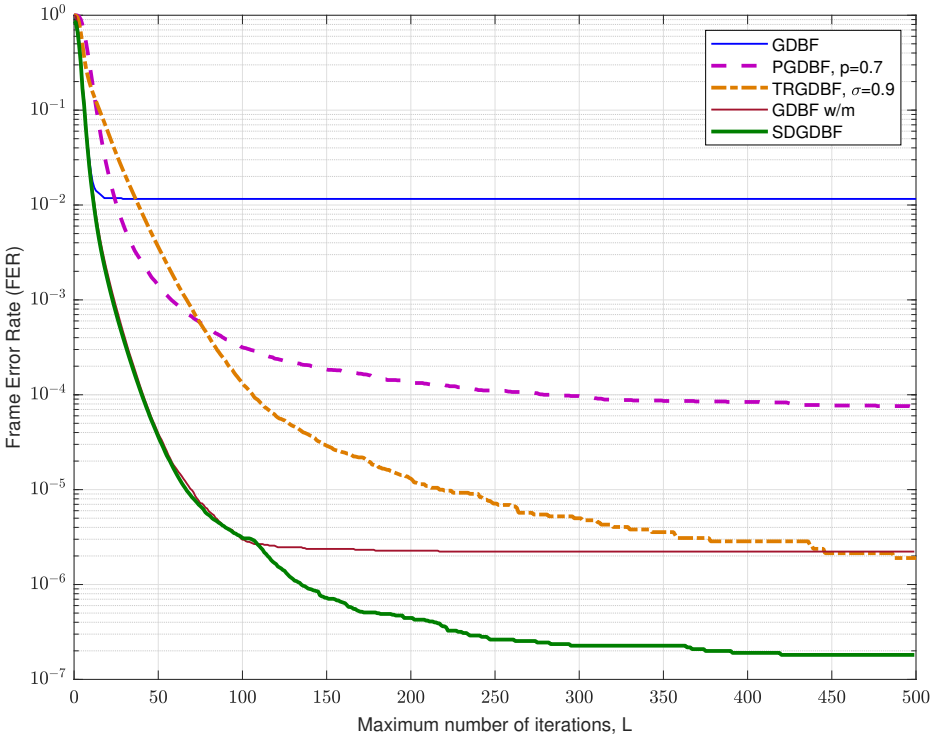


Figure 13. Convergence speed of the various decoding algorithms, QC-LDPC code with $n = 1296$.

Based on its fast convergence, the GDBF-w/m is selected as an optimal candidate for the basic algorithm. The corresponding weighting factors $\alpha = 1$, $\beta = 2$ and momentum vector $\mu = [2, 2, 2, 1]$ are taken from paper [28] (the weighting factors are adjusted to uni-polar codewords, when compared to bi-polar implementation presented in [28], Table I).

As the GDBF-w/m algorithm corrects almost all correctable error patterns in the first 150 iterations, we empirically set the initial value $K_1 = 110$. This value can be reduced

to $K_2 = 50$ if re-initializations are applied to boost the benefit of frequent re-initialization. Every re-initialization is followed with $Z = 1$ modification, so there are approximately 100 iterations in every re-initialization cycle. It can be noticed that the optimal value of K is generally sensitive to BSC crossover probability. For short codes, such as (155, 64) Tanner code, one value of K can be used for a wide range of α_{BSC} without significant performance degradation; however, for longer LDPC codes, it is highly desirable to determine parameter K for any value of α_{BSC}.

Finally, we consider (2640, 1320) Margulis code with the code rate $R = 0.5$ [36], which can be represented by using bipartite graph with $n = 2640$ variable nodes and $m = 1320$ parity checks, with VN degree $\gamma = 3$, CN degree $\rho = 6$, and girth $g = 8$. The corresponding FER performance is presented in Figure 14. It is obvious that the SDGDBF algorithm improves the performance of the GDBF-w/m algorithm (with parameters $\alpha = 1$, $\beta = 2$ and $\mu = [2, 1, 1, 1, 1, 1]$) in the error floor region even for a small value of L. If parameter L can be further increased, the performance is comparable with the floating point decoding algorithms based on the message-passing principle.

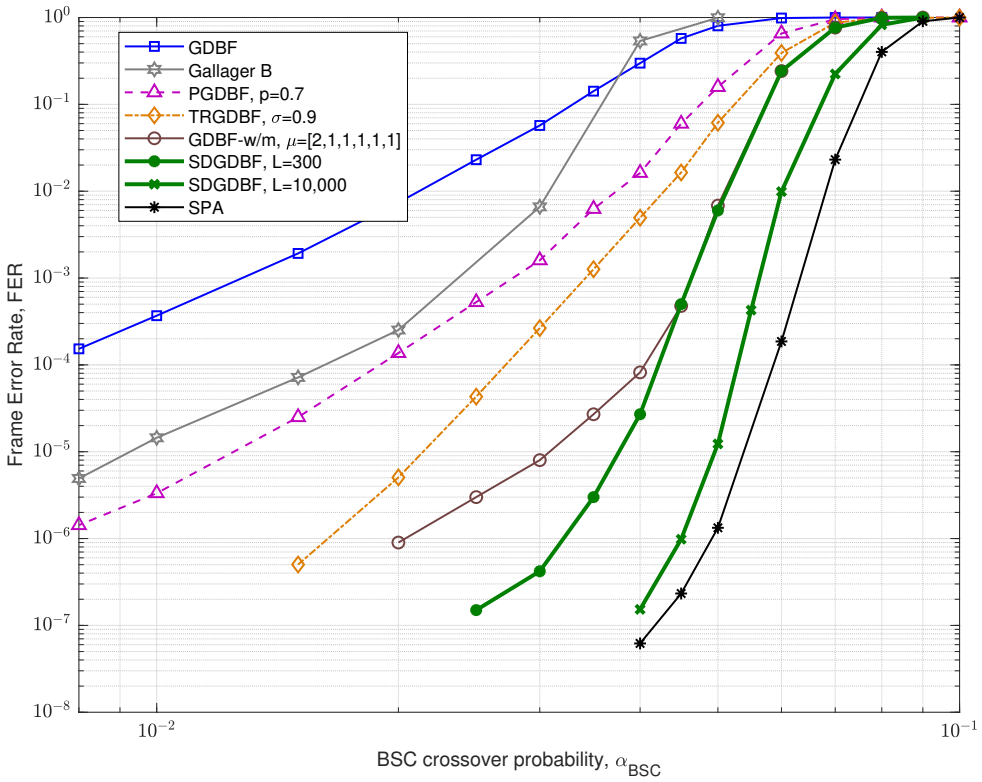


Figure 14. Performance of the various decoding algorithms, Margulis code ($n = 2640$, $\gamma = 3$, $\rho = 6$).

The performance of the analyzed GDBF-based algorithms for $\alpha_{BSC} = 0.03$ and various values of L is presented in Figure 15. It is obvious that the GDBF algorithm with momentum has good performance even for a limited number of decoding iterations. If the modifications presented in Section 3 are applied for $K_1 = 80$, $Z = 1$, the resulting SDGDBF algorithm has the potential to reach the performance of the SPA algorithm after a large number of iterations.

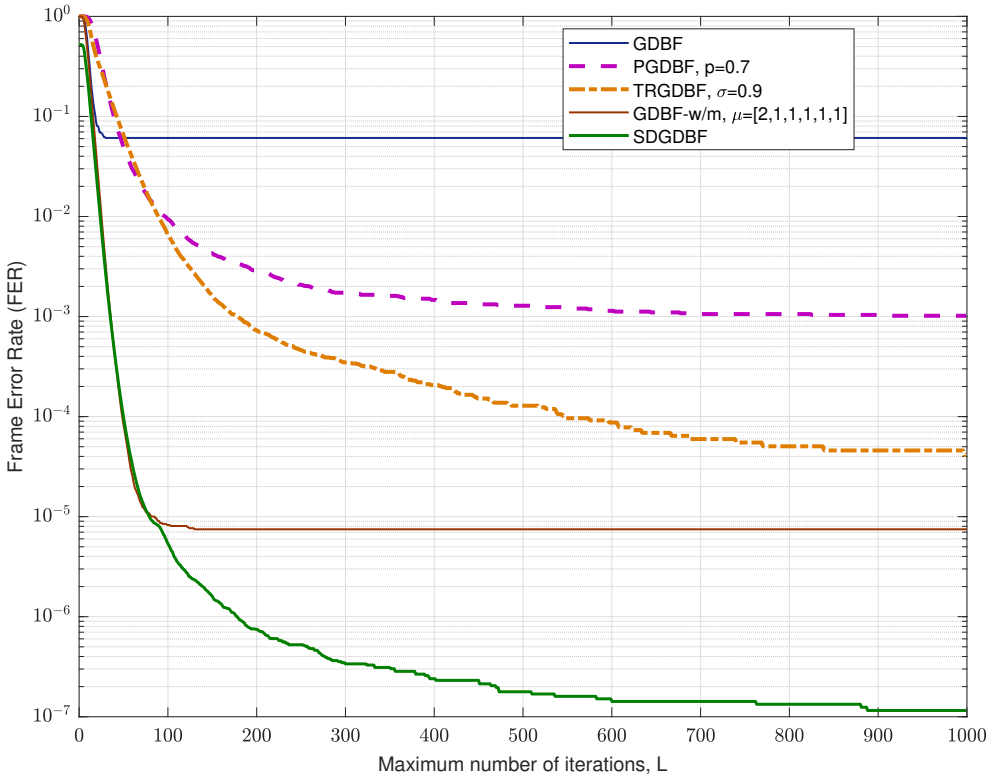


Figure 15. Convergence speed of the various decoding algorithms, Margulis code.

6. Optimization of the Decoder Parameters and Further Work

It has been shown that various deterministic functions can be applied for the re-initialization of the decoder input in multiple decoding attempts. In Figure 16, we present the numerical results that correspond to various decoding strategies, and the results are given for the short Tanner code and $\alpha_{BSC} = 0.01$. The FER plot is given for the classical GDBF algorithm for the BSC channel, i.e., the effect of momentum is not considered in this figure.

In the future work, we will consider the following intriguing directions related to the decoding strategies presented in Sections 3.2 and 3.3:

- The modification described in Algorithm 2 is applied on a codeword estimate after $K = 10$ iterations, i.e., $\mathbf{r} = \Omega(\hat{\mathbf{x}}^{(10)})$, and the procedure is repeated as illustrated in Figure 5a. In this case, the performance improvement is minor, as the Hamming distance between the estimate and the codeword is usually too large after so many iterations. Furthermore, this approach can result in miscorrection (the estimate is equal to another codeword);
- The modification described in Algorithm 2 is applied to the received word, i.e., $\mathbf{r} = \Omega(\mathbf{y})$, and the procedure is repeated as illustrated in Figure 5b. This approach results in a more significant improvement, and miscorrections are avoided. As in the first approach, the FER improvement is most significant for the first modification and diminishes when it is repeated (for a longer code, further improvement can be eventually visible for larger Z);
- On the other hand, we can modify the algorithm to remember a few positions of the flipped bits during the first few iterations of the GDBF algorithm, denoted by $i_1, i_2, i_3, \ldots$. In the q-th decoding attempt, the reference is obtained as $r_{i_q} = y_{i_q} \oplus 1$ and $r_{i_j} = y_{i_j}, j \neq q$. The corresponding transformation $\mathbf{r} = \Phi_q(\mathbf{y})$ results in the flipping of only one bit in the received word. This approach prevents saturating of the FER, as illustrated in Figure 16.

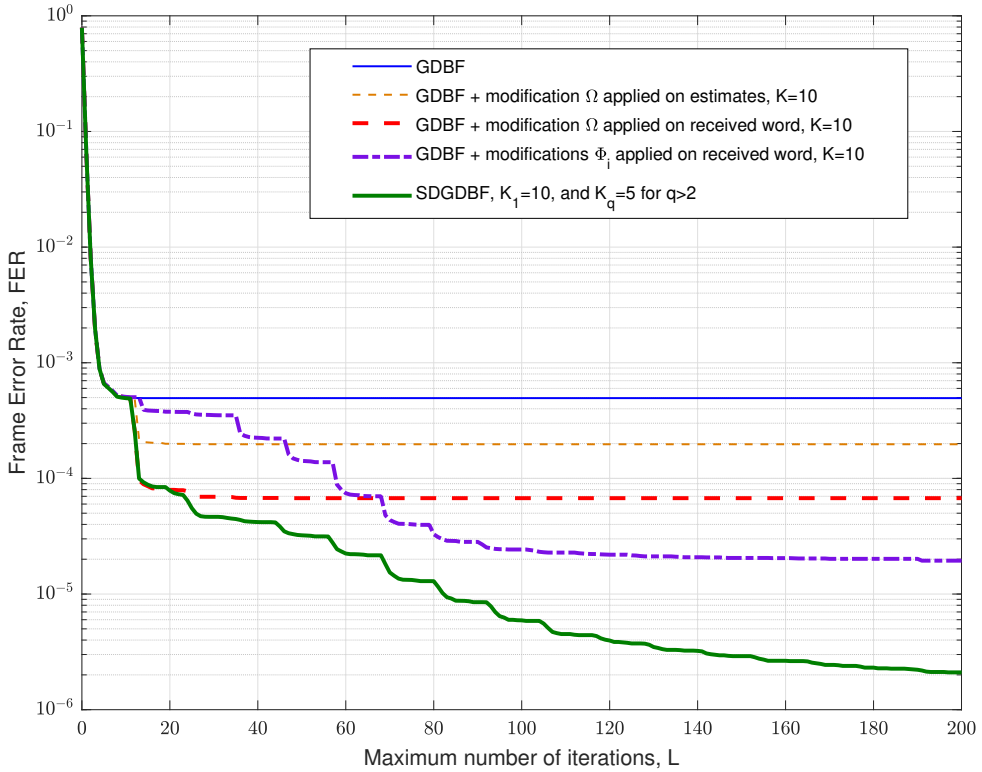


Figure 16. Various strategies for re-initializations, classical GDBF as the base algorithm, Tanner code.

The previously described approaches can be combined in successive decoding attempts for a maximum number of iterations per attempt denoted by K_q. We estimated that the largest performance improvement can be obtained if the proposed modification (given by transformation $\Omega(.)$) is followed by re-initialization based on $\Phi_q(.), \forall q$ (as proposed in the flowchart given in Figure 8). Empirically, we estimate that the GDBF decoder should be initially run for $K_1 = 10$ iterations, and we set $K_q = 5$ for all re-initializations.

In our future research, we will try to further optimize parameters K_q under the condition ($K_1 + K_2 + \cdots = L$), as well as the sequence of applied deterministic functions. In addition, we will concentrate on the extension of the proposed method to the irregular codes, where the variable and check node degrees are node dependent. In such a case, certain conditions (such as in step 2, defined on page 6) will depend on the variable node degree, and therefore the scaling coefficients α and β should be adapted for various variable nodes. We believe that the optimization of these parameters for a specific code by using machine learning techniques will result in the additional improvement of the decoder performance.

7. Discussion and Conclusions

By combining the concept of momentum and the graph properties of the typical trapping sets, we have proposed a deterministic modification of the GDBF algorithm. In previously proposed GDBF-based algorithms, the flipping decisions were based on the energy function that takes into account the number of unsatisfied parity checks connected to the corresponding variable node. In this paper, we analyze some typical error patterns where the variable node is erroneous, but the connected parity checks are satisfied (and vice versa). The proposed modification of the flipping rule takes into account these parity checks, that are no more assumed as valid indicators if the corresponding VN is correct. We have shown that the most significant performance improvement is obtained if the proposed modification is successively applied to the received word. We have also proposed a more general framework based on deterministic re-initializations of the decoder input in multiple decoding attempts.

The proposed algorithm results in significant performance improvement for the LDPC codes with various codeword lengths, obtained by using various construction procedures. It has been shown that the proposed algorithm can be extremely useful for short codes, even for a moderate number of iterations. When the maximum number of iterations can be increased, the proposed algorithm has the potential to provide very reliable transmission even for codes with large codeword lengths.

Author Contributions: Conceptualization, P.I. and S.B.; methodology, P.I.; software, P.I. and B.V.; validation, P.I., S.B. and B.V.; formal analysis, P.I.; investigation, P.I., S.B., and B.V.; writing—original draft preparation, P.I.; writing—review and editing, S.B. and B.V.; visualization, P.I.; supervision, B.V.; project administration, P.I. and B.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Science Fund of the Republic of Serbia, grant No. 6462951, Learning Iterative Decoding Algorithms—LIDA, and the APC was partially funded by National Science Foundation (NSF) under grant CCSS-2027844.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: Predrag Ivaniš and Srdjan Brkić acknowledge the support of the Science Fund of the Republic of Serbia, grants No. 6462951 (Learning Iterative Decoding Algorithms-LIDA) and No 7750284 (Hybrid Integrated Satellite and Terrestrial Access Network—hi-STAR). Bane Vasić acknowledges the support of NSF under grants CIF-1855879, CCF 2106189, CCSS-2027844, CCSS-2052751, and CCF-2100013, as well as the support of the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and funded through JPL's Strategic University Research Partnerships (SURP) program.

Conflicts of Interest: Bane Vasić has disclosed an outside interest in Codelucida to the University of Arizona. Conflict of interest resulting from this interest are being managed by The University of Arizona in accordance with its policies.

References

1. Gallager, R.B. *Low Density Parity Check Codes*; MIT Press: Cambridge, MA, USA, 1963.
2. Richardson, T.; Shokrollahi, M.; Urbanke, R. Design of capacity approaching irregular low-density parity-check codes. *IEEE Trans. Inform. Theory* **2001**, *47*, 619–637. [CrossRef]
3. 3rd Generation Partnership Project. *Technical Specification Group Radio Access Network; NR; Multiplexing and Channel Coding (Release 16), 3GPP TS 38.212 V16.5.0 (2021-03)*; 3GPP: Valbonne, France, 2021.
4. ETSI Digital Video Broadcasting (DVB). *Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications*; Part 2: DVB-S2 Extensions (DVB-S2X), ETSI EN 302307-2 V.1.1.1 (2014-10); ETSI: Sophia Antipolis, France, 2014.
5. *IEEE Standard 802.11-2016*; IEEE Standard for Information Technology—Local and Metropolitan Area Networks—Specific Requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE: New York, NY, USA, 2016.
6. *IEEE Standard 802.16-2004*; IEEE Standard for Local and Metropolitan Area Networks—Part 16: Air Interface for Fixed Broadband Wireless Access Systems. IEEE: New York, NY, USA, 2004.
7. *IEEE Standard 802.3ca-2020*; IEEE Standard for Ethernet Amendment 9: Physical Layer Specifications and Management Parameters for 25 Gb/s and 50 Gb/s Passive Optical Networks. IEEE: New York, NY, USA, 2020.
8. Chen, J.; Fossorier, M.P.C. Near optimum universal belief propagation based decoding of low-density parity check codes. *IEEE Trans. Commun.* **2002**, *50*, 406–414. [CrossRef]
9. Chen, J.; Dholakia, A.; Eleftheriou, E.; Fossorier, M.P.C.; Hu, X.-Y. Reduced-complexity decoding of LDPC codes. *IEEE Trans. Commun.* **2005**, *53*, 1288–1299. [CrossRef]
10. Chen, H.; Abbas, R.; Cheng, P.; Shirvanimoghaddam, M.; Hardjawana, W.; Bao, W.; Li, Y.; Vucetic, B. Ultra-Reliable Low Latency Cellular Networks: Use Cases, Challenges and Approaches. *IEEE Commun. Mag.* **2018**, *56*, 119–125. [CrossRef]
11. Planjery, S.K.; Declercq, D.; Danjean, L.; Vasic, B. Finite Alphabet Iterative Decoders, Part I: Decoding Beyond Belief Propagation on the BSC. *IEEE Trans. Commun.* **2013**, *61*, 4033–4045. [CrossRef]

12. Khoa, L.T. New Direction on Low Complexity Implementation of Probabilistic Gradient Descent Bit-Flipping Decoder. Ph.D. Thesis, Université de Cergy Pontoise, École Nationale Supérieure de l'Électronique et de ses Applications, Cergy Pontoise, France, 2017.
13. Chilappagari, S.K.; Nguyen, D.V.; Vasic, B.; Marcellin, M.W. On Trapping Sets and Guaranteed Error Correction Capability of LDPC Codes and GLDPC Codes. *IEEE Trans. Inf. Theory* **2010**, *56*, 1600–1611. [CrossRef]
14. Wadayama, T.; Nakamura, K.; Yagita, M.; Funahashi, Y.; Usami, S.; Takumi, I. Gradient descent bit flipping algorithms for decoding LDPC codes. *IEEE Trans. Commun.* **2010**, *58*, 1610–1614. [CrossRef]
15. Miladinovic, N.; Fossorier, M. Improved Bit-Flipping decoding of low-density parity-check codes. *IEEE Trans. Inf. Theory* **2005**, *51*, 1594–1606. [CrossRef]
16. Guo, F.; Henzo, H. Reliability ratio based weighted bit-flipping decoding for low-density parity-check codes. *IEEE Electron. Lett.* **2004**, *40*, 1356–1358. [CrossRef]
17. Al Rasheed, O.; Ivaniš, P.; Vasic, B. Fault-Tolerant Probabilistic Gradient-Descent Bit Flipping Decoder. *IEEE Commun. Lett.* **2014**, *18*, 1487–1490. [CrossRef]
18. Sundararajan, G.; Winstead, C.; Boutillon, E. Noisy gradient descent bit-flip decoding for LDPC codes. *IEEE Trans. Commun.* **2014**, *62*, 3385–3400. [CrossRef]
19. Li, H.; Ding, H.; Zheng, L. An escaping scheme for gradient descent bit-flipping decoding of LDPC codes. In Proceedings of the 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Datong, China, 15–17 October 2016.
20. Ivaniš, P.; Al Rasheed, O.; Vasić, B. MUDRI: A Fault-Tolerant Decoding Algorithm. In Proceedings of the 2015 IEEE International Conference on Communications (ICC 2015), London, UK, 8–12 June 2015; pp. 4291–4296.
21. Vasić, B.; Ivaniš, P.; Declercq, D.; LeTrung, K. Approaching Maximum Likelihood Performance of LDPC Codes by Stochastic Resonance in Noisy Iterative Decoders. In Proceedings of the 2016 Information Theory and Applications Workshop (ITA 2016), San Diego, CA, USA, 31 January–5 February 2016.
22. Ivaniš, P.; Vasić, B.; Declercq, D. Performance Evaluation of Faulty Iterative Decoders using Absorbing Markov Chains. In Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT 2016), Barcelona, Spain, 10–15 July 2016.
23. Ivaniš, P.; Vasic, B. Error Errore Eicitur: A Stochastic Resonance Paradigm for Reliable Storage of Information on Unreliable Media. *IEEE Trans. Commun.* **2016**, *64*, 3596–3608. [CrossRef]
24. Declercq, D.; Winstead, C.; Vasic, B.; Ghaffari, F.; Ivanis, P.; Boutillon, E. Noise-Aided Gradient Descent Bit-Flipping Decoders approaching Maximum Likelihood Decoding. In Proceedings of the 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC 2016), Special Session: Noisy Error Correction, Brest, France, 5–9 September 2016.
25. Ren, D.; Sha, J. Improved gradient descent bit flipping decoder for LDPC codes on BSC channel. *IEICE Electron. Express* **2018**, *15*, 20180195. [CrossRef]
26. Cui, H.; Lin, J.; Song, S.; Wang, Z. A New Probabilistic Gradient Descent Bit Flipping Decoder for LDPC Codes. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS 2019), Sapporo, Japan, 26–19 May 2019.
27. Cui, H.; Lin, J.; Wang, Z. An Improved Gradient Descent Bit-Flipping Decoder for LDPC Codes. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2019**, *66*, 3188–3200. [CrossRef]
28. Savin, V. Gradient Descent Bit-Flipping Decoding with Momentum. In Proceedings of the 2021 11th International Symposium on Topics in Coding (ISTC), Montreal, QC, Canada, 30 August–3 September 2021.
29. Le, K.; Ghaffari, F.; Declercq, D.; Vasić, B. Efficient Hardware Implementation of Probabilistic Gradient Descent Bit-Flippings. *Trans. Circuits Syst. I Regul. Pap.* **2017**, *64*, 906–917. [CrossRef]
30. Unal, B.; Akoglu, A.; Ghaffari, F.; Vasić, B. Hardware Implementation and Performance Analysis of Resource Efficient Probabilistic Hard Decision LDPC Decoders. *Trans. Circuits Syst. I Regul. Pap.* **2018**, *65*, 3074–3084. [CrossRef]
31. Jiang, M.; Fan, D. A Low-Latency BF Decoding of LDPC Codes With Dynamic Thresholds. *IEEE Commun. Lett.* **2021**, *25*, 2781–2785. [CrossRef]
32. Le, K.; Declercq, D.; Ghaffari, F.; Spagnol, C.; Popovici, E.; Ivanis, P.; Vasic, B. Efficient realization of probabilistic gradient descent bit flipping decoders. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Lisbon, Portugal, 24–27 May 2015; pp. 1494–1497
33. Le, K.; Declercq, D.; Ghaffari, F.; Kessal, L.; Boncalo, O.; Savin, V. Variable-node-shift based architecture for probabilistic gradient descent bit flipping on QC-LDPC codes. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2018**, *65*, 2183–2195. [CrossRef]
34. Tanner, R.M.; Sridhara, D.; Fuja, T. A class of group-structured LDPC codes. In Proceedings of the ISTA, Ambleside, UK, 20 March 2001.
35. Tanner, R.M.; Sridhara, D.; Sridharan, A.; Fuja, T.; Costello, D.J. LDPC block and convolutional codes based on circulant matrices. *IEEE Trans. Inf. Theory* **2004**, *50*, 2966–2984. [CrossRef]
36. Margulis, G.A. Explicit constructions of graphs without short cycles and low density codes. *Combinatorica* **1982**, *2*, 71–78. [CrossRef]