Digital Communications and Networks 10 (2024) 217–226

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

Doped low-density parity-check codes

Yong Li [a,*], Rui Liu [a], Xianlong Jiao [a], Youqiang Hu [b,**], Zhen Luo [c], Francis C.M. Lau [d]

[a] College of Computer Science, Chongqing University, Chongqing, 400044, China
[b] School of Automation, Chongqing University, Chongqing, 400044, China
[c] College of Electronic and Information Engineering, Southwest University, Chongqing, 400715, China
[d] Future Wireless Networks and IoT Focusing Area, Department of Electronic and Information Engineering, Hong Kong Polytechnic University, Hong Kong, China

ARTICLE INFO

Keywords:
LDPC codes
Doped LDPC codes
Tanner graph
Quadratic residue codes
Trapping sets

ABSTRACT

In this paper, we propose a doping approach to lower the error floor of Low-Density Parity-Check (LDPC) codes. The doping component is a short block code in which the information bits are selected from the coded bits of the dominant trapping sets of the LDPC code. Accordingly, an algorithm for selecting the information bits of the short code is proposed, and a specific two-stage decoding algorithm is presented. Simulation results demonstrate that the proposed doped LDPC code achieves up to 2.0 dB gain compared with the original LDPC code at a frame error rate of 10^{-6}. Furthermore, the proposed design can lower the error floor of original LDPC codes.

1. Introduction

Since their rediscovery in the 1990s [1], LDPC codes have attracted researchers' attention due to their impressive capacity-approaching capabilities and practical decoding algorithms. So far, they have been adopted in several industrial standards such as wireless local area network (IEEE 802.11n) [2], DVB-S2 for video broadcast via satellite, WiMAX(IEEE 802.16e) [3], IEEE 802.3an for 10 Gbit/s Ethernet [4], and the fifth generation mobile communication system (5G) [5]. Although typical short and moderate-length (say $\leq$2000 bits) LDPC codes for wireless and wireline applications have demonstrated excellent waterfall performance down to a Bit Error Rate (BER) level of 10^{-7} even to 10^{-10}, they often suffer from an error floor, especially at high Signal-to-Noise Ratio (SNR). That is, the error performance of short and moderate-length LDPC codes ceases to improve much despite using higher SNR channels. Consequently, they are not very suitable for those high-throughput, power-efficient applications such as optical communication and flash memory, where the requirements on data reliability are typically under a Frame Error Rate (FER) of 10^{-12}.

The error floor problem of an LDPC code is attributed to some topological substructures of the code's Tanner graph. For the Binary Erasure Channel (BEC), the graphical structures related to the error floor are known as stopping sets. A stopping set $\mathcal{S}$ is a variable-node set in which all neighbors of $\mathcal{S}$ are connected to $\mathcal{S}$ at least twice. As a result, the variable nodes cannot be determined by the iterative decoder even if all other bits are known. It is because every neighbor of $\mathcal{S}$ has at least two connections to this set, and thus all messages to $\mathcal{S}$ are erased messages [6]. For the Additive White Gaussian Noise (AWGN) channel, the graphical structures associated with the error floor are called trapping sets [7], which are defined as a specific variable-node set that is connected to some odd-degree check nodes. Trapping sets are characterized by a pair of parameters (a, b), where a is the number of variable nodes, and b is the number of check nodes with odd degrees. Trapping sets which are closely related to the error floor are called dominant trapping sets whose (a, b) parameters are typically small. For the Binary Symmetric Channel (BSC), the graphical structures which limit the performance of iterative decoders are called absorbing sets. Absorbing sets have a similar definition as trapping sets and also control the error floor behavior of message-passing decoders [8].

Over the past two decades, much work has been devoted to lowering the error floor. It can be roughly divided into two categories: (1) code construction avoiding certain topological structures and (2) modification of decoding schemes. For example, Asvadi et al. [9] constructed Quasi-Cyclic (QC) LDPC codes with low error floors by eliminating the dominant trapping sets of the base code. In Ref. [10], Nguyen et al. designed structured regular LDPC codes with low error floors over the Binary Symmetric Channel (BSC) by avoiding certain small trapping sets. Recently, Wang et al. [11] designed separable circulant-based LDPC

* Corresponding author.
** Corresponding author.
E-mail addresses: yongli@cqu.edu.cn (Y. Li), liurui_cs@cqu.edu.cn (R. Liu), xljiao@cqu.edu.cn (X. Jiao), yqhu@cqu.edu.cn (Y. Hu), luozhenluozhen@gmail.com (Z. Luo), francis-cm.lau@polyu.edu.hk (F.C.M. Lau).

https://doi.org/10.1016/j.dcan.2022.11.013
Received 12 February 2022; Received in revised form 27 November 2022; Accepted 28 November 2022
Available online 2 December 2022

codes free of certain absorbing sets by utilizing the cycle consistency matrix. Tao et al. [12] presented a construction of QC-LDPC codes with degree-3 variable nodes and girth 8 based on fully-connected protographs, where small elementary trapping sets were removed by avoiding certain cycles of length 8 in the Tanner graph. In Ref. [13], Sarıduman et al. developed a simulated-annealing-based method to design regular QC-LDPC codes without dominant trapping sets and thus with good performance in the error floor region. Most recently, Karimi et al. [14] presented a systematic design of protograph-based QC-LDPC codes with low error floors by characterizing the trapping sets of the codes and eliminating a targeted collection of trapping sets. Moreover, researchers proposed Generalized LDPC (GLDPC) codes [15,16], which are better than original LDPC codes in the error floor region by replacing the single parity checks with generalized constraints. However, such a construction leads to a noticeable rate loss which is unacceptable in those scenarios requiring high bandwidth efficiency.

Alternatively, different decoding strategies have been proposed to lower the error floor. Han et al. [17] designed three low-error-floor decoders by identifying the trapping set information of an LDPC code based on simulation or graph-search techniques. Zhang et al. [18] designed a high-throughput parallel-serial decoder architecture and demonstrated that dually-quantized sum-product decoders and approximate sum-product decoders lower the error floor of LDPC codes by alleviating the effects of low-weight absorbing sets. In Ref. [19], Casado et al. presented Informed Dynamic Scheduling (IDS) strategies that select the messages to update, converge in fewer iterations, and produce a lower error floor than either flooding or sequential scheduling. Angarita et al. [20] developed a reduced-complexity Min-Sum (MS) algorithm and showed that such an algorithm achieves better error-floor performance when compared with the normalized MS algorithm. Li et al. [21] proposed a low-complexity post-processing scheme to significantly lower the error floor of an MS-based LDPC decoder by introducing perturbations and hence helping the decoder escape from undesired local maximums. Lee et al. [22] developed a low-complexity redecoding-based error-floor lowering technique for QC-LDPC codes, where a predetermined set of variable nodes, obtained by using a two-stage off-line search, are attenuated before the redecoding. Tao et al. [23] were inspired by simulated annealing and generalized the post-processor design, which lowers the error floors by two orders of magnitude for two LDPC codes. Based on the assumption that trapping sets are the primary cause of error floor for quantized LDPC decoders, Hatami et al. [24] presented a modified MS algorithm which outperforms the conventional MS algorithm, the attenuated MS algorithm, and the offset MS algorithm across all SNRs.

Although the above-mentioned construction methods and decoding schemes were developed to lower the error floor, an effective and low-complexity strategy to design an LDPC code with good performance (including code construction and decoding strategy) in both waterfall and error-floor regions is still lacking. In this paper, we first present a doping method to lower the error floor of LDPC codes by integrating the Tanner graphs of an LDPC code and a short block code as a whole [26]. Herein, all the information bits of the short block code are chosen from the coded bits of the LDPC code. The resultant codes are referred to as doped LDPC codes thereafter, where the LDPC code and the short block code are called the global code and the component code, respectively. Actually, our work can be seen as a generalization of [27], in which additional single parity checks were introduced to lower the error floors of LDPC codes. In our framework, an algorithm for selecting information bits of the component codes is developed by utilizing the information of dominant trapping sets of LDPC codes which can be efficiently obtained according to the algorithm given in Ref. [28]. (Therein, the influence of each type of trapping set on the code performance is measured based on the method given in Ref. [29].) We expect that those variable nodes forming the dominant trapping sets of the LDPC code are selected to encode the component code. The objective is to provide extra "external" information to the variable nodes in the trapping sets and hence help them converge to the correct bit values.

It is worth pointing out that those short block codes with low to high rates can be employed as component codes in our framework. Since they are much shorter than the original LDPC codes, the resultant rate loss is very small. On the contrary, high-rate component codes are required for GLDPC codes due to significant rate loss resulting from replacing the single parity checks with super checks. Here Quadratic Residue (QR) codes are considered as the component codes in our simulations, but other linear block codes are also applicable.

In order to decode the proposed doped LDPC codes, a two-stage decoding algorithm is further developed. In the first stage, the conventional log-domain Belief Propagation (log-BP) algorithm and the A Posteriori Probability (APP) decoding algorithm given in Ref. [25] are utilized (it is called the one-stage decoding algorithm hereafter). If no valid codeword is output, the second-stage decoding will start. During this stage, a variable γ is introduced to measure the decoding reliability of the component codes after the first-stage decoding. Subsequently, two submodes are presented to finish the second-stage decoding in terms of the value of γ.

Simulation results show that the proposed doped LDPC codes with the one-stage decoding algorithm achieve 1.6 dB and 1.4 dB gains, respectively, at a FER of 10^{-5}, when the (204, 102) and the (408, 204) Mackay codes are considered. Nevertheless, the doped LDPC codes with the one-stage decoding algorithm provide no gain when the (672, 336) Margulis code and the (279, 158) RS-LDPC code are tested. We also note that the doped LDPC codes always perform better than the original LDPC codes if the two-stage decoding algorithm is utilized and the two-stage algorithm provides noticeable gains over the one-stage decoding algorithm in the four examples. Furthermore, the rate loss due to doping is small when the doped LDPC codes are constructed by long LDPC codes and short block codes. Complexity analysis demonstrates that almost the same average number of iterations is required for both the one-stage decoding algorithm and the original log-BP algorithm, while the CPU runtime of the one-stage decoding algorithm is slightly longer than that of the log-BP algorithm. Moreover, although the two-stage algorithm requires much more CPU runtime than the one-stage algorithm in the low SNR regimes, both of them use comparable average decoding time in the high SNR regimes.

It should be highlighted that the performance of the proposed doped LDPC code is closely related to the component code. It is because the information length of the component code represents the number of Variable Nodes (VNs) located in the trapping sets of the original LDPC code that can receive the extrinsic information from the Check Nodes (CNs) of the component code. One intuition is that longer component codes are required to provide large gains if longer LDPC codes (say with 1000, 2000 bits) are chosen to construct the doped LDPC codes. Nevertheless, we believe that the proposed construction of doped LDPC codes, which consists of the algorithm of selecting the information bits of the component code and the doping method, is effective in designing good doped LDPC codes.

The remainder of this article is organized as follows. The concept of doped LDPC codes and the (traditional) one-stage decoding algorithm are first introduced in Section 2, and this section also reviews the knowledge of trapping sets. Section 3 presents the proposed novel design of doped LDPC codes and the method to select the information bits of the component code based on the trapping-set concept. Section 4 describes the two-stage decoding algorithm. Simulation results and complexity analysis are discussed in Section 5 and Section 6, respectively. Finally, Section 7 concludes this paper with a brief summary.

2. Terminology and background of doped LDPC codes and trapping sets

2.1. Doped LDPC codes

We assume that a doped LDPC code is constructed by adding the check bits of a short block code after the coded bits of an LDPC code, where the information bits of the component code are selected from the LDPC codeword. We denote N and M as the code length and information length of the LDPC code and n and m as the code length and information length of the component code, respectively. Then the corresponding Tanner graph contains N+(n−m) variable nodes, (N−M) Single Parity Check (SPC) nodes, and one generalized check node called a Component Check (CC) node in the sequel. Fig. 1 shows an example of the doped LDPC code, which consists of an LDPC code (for illustration purpose) and a component code. In this example, v_1, v_2, …, v_8 correspond to an LDPC codeword, and (v_2, v_4, v_6, v_8, v_9, v_{10}, v_{11}) represents a component codeword in which v_2, v_4, v_6, v_8 are the information bits. Moreover, the component code is represented by a Component Check (CC) node (large square), some extra VNs, and extra connections in the Tanner graph of the original LDPC code. The corresponding code rate of the resultant doped LDPC code is then given by

$$R \geq \frac{M}{N+n-m} \tag{1}$$

In this paper, we denote x as a scalar, $\boldsymbol{x}$ as a vector, $\mathcal{A}$ as a set, and $\mathbb{A}$ as a matrix. The one-stage decoding algorithm of the doped LDPC codes is similar to the log-BP algorithm for conventional LDPC codes. We denote by $r^t_{j\to i}$ (respectively $q^t_{i\to j}$) the message passed from check node j (respectively variable node i) to variable node i (respectively check node j) for the original LDPC code, where the superscript t indicates the iteration number, $i = 1, \ldots, N, j = 1, \ldots, N-M$. Let L($P_k$), $k = 1, \ldots, N+n-m$ be the log-likelihood channel message of the k-th received bit. Then the decoding steps at each iteration are summarized as follows.

1) Update $r^t_{j\to i}$

Considering the SPC nodes of a doped LDPC code, the update formula of $r^t_{j\to i}$ (j = 1, …, N−M) is the same as that in the log-BP algorithm for decoding LDPC codes, i.e.,

$$r^t_{j\to i} = 2\tanh^{-1}\left(\prod_{i'\in\mathcal{N}(j)\backslash i} \tanh\left(\frac{1}{2}q^{t-1}_{i'\to j}\right)\right) \tag{2}$$

where $\mathcal{N}(j)$ is the set of Variable Nodes (VNs) connected to the check node j.

In a doped LDPC code, we assume that there is only one component code and denote it by s. We use the APP algorithm given in Ref. [25] to compute the message transmitted from the CC node to its neighboring variable nodes. The messages of the CC node received from the neighboring variable nodes in the previous iteration are denoted as

$$\boldsymbol{q}^{t-1}_s = [q^{t-1}_{g_1\to s}, \ldots, q^{t-1}_{g_n\to s}] \tag{3}$$

where $\mathcal{N}(s) = \{g_1, g_2, \ldots, g_n\}$ represents the set of neighboring VNs of the component code s. Note that g_1, g_2, …, g_m are also VNs of the original LDPC code. The APP algorithm computes the log-likelihood values which are denoted as

$$\mathbf{y}^t_s = [y^t_{s\to g_1}, \ldots, y^t_{s\to g_n}] \tag{4}$$

Then the extrinsic messages passed from the CC node to neighboring variable nodes are given by

$$\mathbf{u}^t_s = [u^t_{s\to g_1}, \ldots u^t_{s\to g_n}] \tag{5}$$

The relation among $\mathbf{u}^t_s$, $\mathbf{y}^t_s$ and $\mathbf{q}^{t-1}_s$ is

$$\mathbf{u}^t_s = \mathbf{y}^t_s - \mathbf{q}^{t-1}_s \tag{6}$$

2) Update $q^t_{i\to j}$ for i = 1, 2, …, N

$$q^t_{i\to j} = \begin{cases} L(P_i) + \sum_{j'\in\mathcal{C}(i)\backslash j} r^t_{j'\to i} + u^t_{s\to i} & \text{if } i \in \mathcal{N}(s) \\ L(P_i) + \sum_{j'\in\mathcal{C}(i)\backslash j} r^t_{j'\to i} & \text{otherwise} \end{cases} \tag{7}$$

where $\mathcal{C}(i)$ is the set of Check Nodes (CNs) connected to the variable node i.

Update $q^t_{i\to s}$ for $i \in \mathcal{N}(s)$

$$q^t_{i\to s} = L(P_i) + \sum_{j'\in\mathcal{C}(i)} r^t_{j'\to i} \tag{8}$$

3) Update the posterior log-likelihood values

$$L(Q_i) = \begin{cases} L(P_i) + \sum_{j'\in\mathcal{C}(i)} r^t_{j'\to i} + u^t_{s\to i} & \text{if } i \in \mathcal{N}(s) \\ L(P_i) + \sum_{j'\in\mathcal{C}(i)} r^t_{j'\to i} & \text{otherwise} \end{cases} \tag{9}$$

4) Hard decision

$$\hat{v}_i = \begin{cases} 0, & \text{if } L(Q_i) > 0 \\ 1, & \text{otherwise} \end{cases} i = 1, \ldots, N \tag{10}$$

If the maximum iteration number is reached or the equation $\mathbf{H}\cdot\hat{\boldsymbol{v}}^T = \mathbf{0}$ ($\hat{\boldsymbol{v}} = [\hat{v}_1, \ldots, \hat{v}_N]$) is satisfied where $\mathbf{H}$ is the parity check matrix of an LDPC code, then the algorithm terminates. Otherwise, it continues with the next iteration.

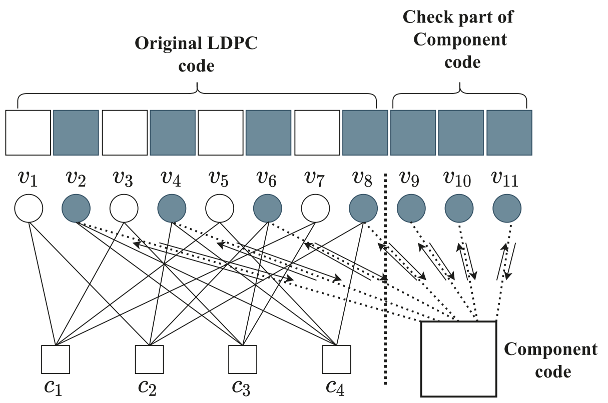

Original LDPC code
Check part of Component code
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}
c_1 c_2 c_3 c_4
Component code

Fig. 1. The architecture and the Tanner graph of a doped LDPC code.

2.2. *Trapping sets*

The term trapping set first proposed by Richardson [7] was defined as a specific set of VNs connected to some CNs with odd degrees. An (a, b) trapping set is a configuration of a VNs, which induces a subgraph including b unsatisfied CNs (i.e., odd-degree CNs) and any possible number of satisfied CNs (i.e., even-degree CNs). In a trapping set, the satisfied CNs contribute to the error floor of an LDPC code because they tend to keep the current hard-decision output of the decoder that corresponds to an incorrect codeword. On the other hand, unsatisfied CNs can pass the correct extrinsic message into the trapping set and help the trapped VNs escape from the trapping set.

A trapping set is called an (a, b) elementary trapping set if there are a VNs inducing b unsatisfied CNs whose degrees are only one or two in the induced subgraph S. In the literature, elementary trapping sets are known to be the dominant contributor to the error floor [30]. Define $G = (V \cup C$, $E)$ as the bipartite graph of an LDPC code with the set of VNs $V = \{v_1, v_2$, …, $v_N\}$, the set of CNs $C = \{c_1, c_2$, …, $c_{N-M}\}$, and the edge set E. Let S be the subgraph induced by an (a, b) trapping set contained in G and V_S be the set of VNs in S. Further, denote C_S as the set of CNs adjacent to the VNs in V_S. Let $C_1 \subseteq C_S$ be the set of unsatisfied CNs with degree-one in the subgraph S and $V_1 \subseteq V_S$ be the set of neighboring VNs of those CNs in C_1. For the convenience of algorithm description in the following sections, two concepts, i.e., the computation tree and the separation [31,32], are introduced as follows.

Definition 1 (*k-iteration computation tree*): A *k-iteration* computation tree for a message-passing decoder in G, denoted by $T_k(v)$, is a tree constructed by selecting a variable node $v \in V$ as its root and then recursively adding the edges and leaf nodes according to the message-passing path during k decoding iterations.

Definition 2 (*k-separation*): Given a Tanner graph G and a subgraph S induced by a trapping set, a variable node $v \in V_1$ is said to be *k-separated* if there exists at least one neighboring degree-one check node, no descendant of which is included in V_S. If every VN in V_1 is k-separated, we say that the induced subgraph S satisfies the k-separation assumption.

In Fig. 2, we show the graph of a (5, 1) trapping set in the (96, 48) Mackay code and some of its neighboring nodes. The set of VNs in the trapping set is $V_S = \{v_1, v_2, v_3, v_4, v_5\}$ indicated by solid black circles, and the set of CNs in the trapping set is $C_S = \{c_i\}$, $1 \le i \le 8$. In this trapping set, only C_8 is the degree-one CN, i.e., $V_1 = \{v_4\}$ and $C_1 = \{c_8\}$. Accordingly, v_6, v_7 are the neighboring VNs of the trapping set. Fig. 3 illustrates a computation tree with the root v_4 of the subgraph induced by the above (5, 1) trapping set. According to the above definition, the VN v_4 has the separation of 1 since there exists no leaf nodes belonging to V_S in $T_1(v_4)$ while the descendants v_1 and v_5 belong to V_S in $T_2(v_4)$. A theorem about the k-separation is given below.

Theorem 1. (*Theorem* 4, [32]): *Let G be the Tanner graph of a variable-regular LDPC code that contains an induced subgraph S of a trapping set. Suppose that the channel is either a Binary Symmetric Channel* (*BSC*) *or an Additive White Gaussian Noise* (*AWGN*) *channel, and that the message transmitted from the channel to all VNs outside the subgraph S are correct. If S satisfies the k-separation assumption for sufficiently large k, then all erroneous VNs in S can be corrected successfully with the BP-based decoder.*

3. The proposed design of doped LDPC codes

In the conventional GLDPC configuration, the performance improvement is at the cost of an evident rate loss, which hinders its applications in those high-rate scenarios. In this paper, a new design algorithm of doped LDPC codes is proposed, in which some VNs of an LDPC code are selected as the VNs of the component code, i.e., as the information bits of the component code. In other words, a doped LDPC codeword can be generated by re-encoding some selected coded bits of an LDPC codeword and concatenating the LDPC codeword and the check bits of the component code. The corresponding Tanner graph is given in Fig. 1. In such a configuration, some VNs of the original LDPC code and all the VNs corresponding to the check bits of the component code are connected to a CC node. As shown in Fig. 1, nodes v_2, v_4, v_6, v_8, v_9, v_{10} and v_{11} are connected to a CC node and thus satisfy the constraints of the component code.

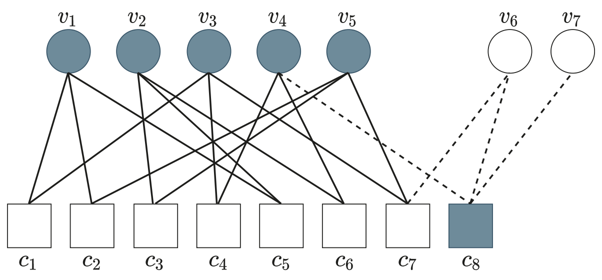


Fig. 2. The Tanner graph of a (5, 1) trapping set and its neighboring VNs (v_6, v_7) in the (96, 48) Mackay code.

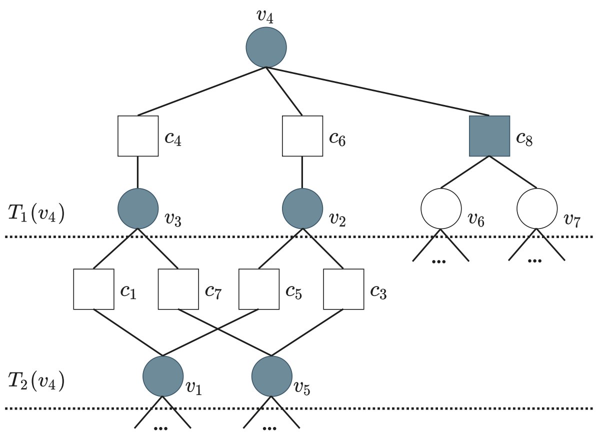
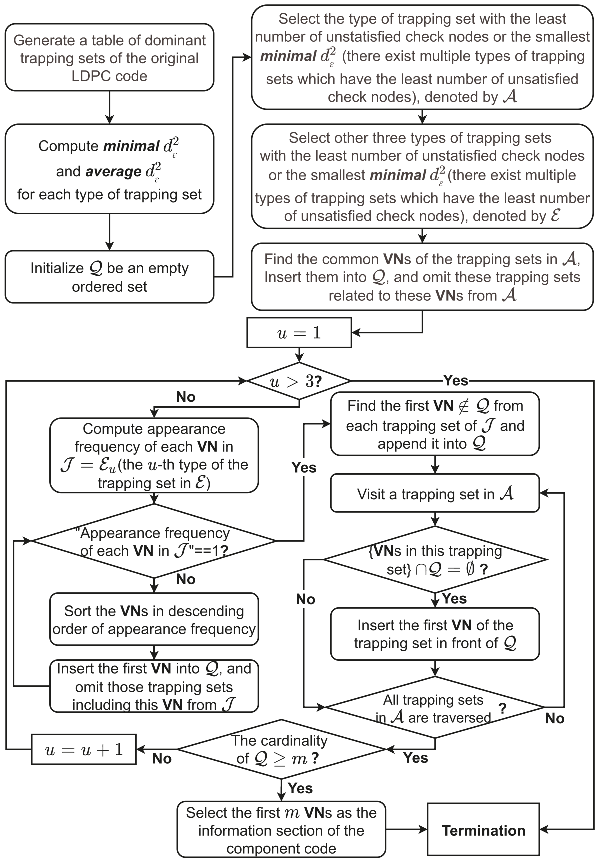


Fig. 3. The computation tree with root v_4.

Proposition 1. *Let us select one or a few VNs from an* (a, b) *elementary trapping set as VNs of the component code. Assume that all VNs of the component code are correct, or that they can always be corrected after the APP decoding. Then the* (a, b) *trapping set will be reduced to a smaller one or even disappear.*

Proof: The proof is similar to that of Theorem 1 in Ref. [32]. If a VN v_r in a trapping set Γ is connected to a CC node, this VN can escape from Γ with the aid of the CC node according to the assumption. By removing this VN and its associated edges from the subgraph induced by Γ, a smaller elementary trapping set Γ' is formed because some satisfied CNs become unsatisfied CNs. These new unsatisfied CNs will further increase the separation degree of their neighboring VNs (a k-separated VN is said to have a separation degree of k) with a high probability and thus make



Generate a table of dominant trapping sets of the original LDPC code
Compute ***minimal*** d_ε^2 and ***average*** d_ε^2 for each type of trapping set
Initialize $\mathcal{Q}$ be an empty ordered set
Select the type of trapping set with the least number of unstatisfied check nodes or the smallest ***minimal*** d_ε^2 (there exist multiple types of trapping sets which have the least number of unsatisfied check nodes), denoted by $\mathcal{A}$
Select other three types of trapping sets with the least number of unstatisfied check nodes or the smallest ***minimal*** d_ε^2 (there exist multiple types of trapping sets which have the least number of unsatisfied check nodes), denoted by $\mathcal{E}$
Find the common **VN**s of the trapping sets in $\mathcal{A}$, Insert them into $\mathcal{Q}$, and omit these trapping sets related to these **VN**s from $\mathcal{A}$
$u = 1$
$u > 3$**?**
Yes
No
Compute appearance frequency of each **VN** in $\mathcal{J} = \mathcal{E}_u$(the u-th type of the trapping set in $\mathcal{E}$)
"Appearance frequency of each **VN** in $\mathcal{J}$"==1**?**
Yes
No
Sort the **VN**s in descending order of appearance frequency
Insert the first **VN** into $\mathcal{Q}$, and omit those trapping sets including this **VN** from $\mathcal{J}$
Find the first **VN** $\notin \mathcal{Q}$ from each trapping set of $\mathcal{J}$ and append it into $\mathcal{Q}$
Visit a trapping set in $\mathcal{A}$
{**VN**s in this trapping set} $\cap \mathcal{Q} = \emptyset$ **?**
No
Yes
Insert the first **VN** of the trapping set in front of $\mathcal{Q}$
All trapping sets in $\mathcal{A}$ are traversed **?**
No
Yes
$u = u + 1$
No
The cardinality of $\mathcal{Q} \ge m$ **?**
Yes
Select the first m **VN**s as the information section of the component code
Termination

Fig. 4. The flowchart of **Algorithm** 1.

erroneous VNs be corrected for a sufficiently large number of iterations. Consequently, the original trapping set Γ will shrink or even disappear.

In summary, a VN v_r of an LDPC code will escape from a trapping set with a high probability after being connected to a CC node and will further generate new unsatisfied CNs in the corresponding trapping set. Accordingly, these unsatisfied CNs increase the separation of the neighboring VNs and hence help the VNs escape from trapping sets as the iteration number increases. Therefore, the design given in Fig. 1 eliminates or mitigates the trapping set, thereby improving the performance of the LDPC code. In this paper, we choose QR codes as the component code for illustration, although other short systematic block codes can also be used in the proposed doped LDPC framework. In practice, this type of doped LDPC codes can be viewed as a class of two-edge type LDPC code. One edge-type corresponds to the LDPC code, and the other is related to the component code.

It can readily be understood that selecting VNs of the component code directly influences the performance of doped LDPC codes. Yuan et al. [27] developed a bit-searching algorithm in which the most erroneous VNs and the most reliable VNs are first located based on Monte Carlo simulations. Then new CNs connecting the most erroneous VNs and the most reliable VNs are added to lower the error floor. (However, this algorithm does not utilize the knowledge of trapping sets of an LDPC code.) The bit-searching algorithm given in Ref. [27] has a drawback: most of the chosen VNs may belong to one or two trapping sets, whereas some chosen VNs are not included in any trapping sets, leading to the fact that there exists no edge between some harmful trapping sets and those newly added CNs. Towards this end, we develop a bit-selecting algorithm, called **Algorithm** 1, based on the knowledge of trapping sets, as follows. The corresponding flowchart is illustrated in Fig. 4.

Algorithm 1 Bit-Selecting Algorithm

```
1: Generate a table of dominant trapping sets of the original LDPC code by using the algorithm given in [28], in which trapping sets are classified in terms of different values of $a$ and $b$. Herein, two parameters, minimal $d_\varepsilon^2$ and average $d_\varepsilon^2$ of each type of trapping set are computed, where $d_\varepsilon^2$ refers to the squared Euclidean distance to the error boundary [29]. Further, let $Q$ be an empty ordered set.
2: Select the type of trapping set with the least number of unsatisfied check nodes (choose the one with the smallest minimal $d_\varepsilon^2$ in case the desired trapping set is not unique) from the above table and denote it by $\mathcal{A}$.
3: Choose three types of trapping sets with the least number of unsatisfied check nodes or smallest minimal $d_\varepsilon^2$ (when the number of unsatisfied check nodes is the same) from the above table excluding $\mathcal{A}$ and form a new group of trapping sets, denoted by $\mathcal{E}$.
4: If some trapping sets contained in $\mathcal{A}$ have common VNs, insert these VNs into $Q$ and omit the corresponding trapping sets from $\mathcal{A}$. Then set $u = 1$.
5: If $u$ is larger than 3, then go to Step 8). Define $\mathcal{J} = \mathcal{E}_u$, where $\mathcal{E}_u$ denotes the $u$-th type of trapping set in $\mathcal{E}$. We count the appearance frequency of each VN in $\mathcal{J}$ and sort the VNs in descending order of appearance frequency. We insert the first VN in the arranged sequence into $Q$ and omit those trapping sets including this VN from $\mathcal{J}$. Repeat such a process until the appearance frequency of each VN belonging to $\mathcal{J}$ is equal to 1. Then find the first VN not belonging to $Q$ from each trapping set of $\mathcal{J}$ and insert it into $Q$ (always insert these VNs at the end of $Q$.).
6: Traverse all the trapping sets in $\mathcal{A}$. If there exists no common VN between a trapping set and $Q$, insert the first VN of the trapping set in front of $Q$.
7: If the cardinality of $Q$ is larger than or equal to $m$, select the first $m$ VNs as the information section of the component code, and then go to Step 8). Otherwise, set $u = u + 1$ and go to Step 5).
8: Algorithm termination.
```

Remarks: 1) In the second step of **Algorithm** 1, we first select a type of trapping set with the least number of unsatisfied CNs and denote it by $\mathcal{A}$. It is very likely that this type of trapping set contributes to the error floor. The VNs in $\mathcal{A}$ should be considered with the highest priority when selecting the VNs of the component code.

2) We guarantee that the selected VNs are associated with as many trapping sets in $\mathcal{A}$ as possible. If a sufficient number of VNs (i.e., m VNs) cannot be selected from the trapping sets in $\mathcal{A}$, other VNs in other types of trapping sets not belonging to $\mathcal{A}$ will be considered (i.e., $\mathcal{E}$). In selecting VNs, we always follow the priority given in **Algorithm** 1 and cover as many dominant trapping sets as possible.

3) If three types of trapping sets cannot provide enough VNs needed to encode the component code, four or more types of trapping sets will be selected to form $\mathcal{E}$ in the third step of **Algorithm** 1.

4) Experimental results show that the trapping sets in $\mathcal{A}$ have a larger effect on the decoding performance than those in $\mathcal{E}$. Therefore, we always insert those VNs from $\mathcal{A}$ into the left-most position of $\mathcal{Q}$ but insert the ones from $\mathcal{E}$ at the end of $\mathcal{Q}$ (Note that $\mathcal{Q}$ is an ordered set). The arrangement ensures that the VNs in $\mathcal{A}$ have more chance to be selected than the ones in $\mathcal{E}$ since we always select the first m VNs of $\mathcal{Q}$ to encode the component code.

5) Only one VN of each trapping set in $\mathcal{A}$ is inserted into $\mathcal{Q}$ since correcting an erroneous VN often helps other VNs escape from the trapping set. This method increases the variety of trapping sets; thus, more VNs belonging to different trapping sets can receive the extrinsic information from the component code.

In order to explain **Algorithm** 1 more clearly, we take the (204, 102) Mackay code as an example.

Step 1 The table of dominant trapping sets of this code is generated as shown in Table 1 in Section 5.

Step 2 The unique (4, 0)-type trapping set consisting of VNs 114, 154, 164 and 199 is selected, i.e., $\mathcal{A} = \{\{114, 154, 164, 199\}\}$.

Step 3 The (5, 1)-, (9, 1)- and (7, 1)-types of trapping sets are selected and form $\mathcal{E}$. As illustrated in Table 1, $\mathcal{E}$ consists of 39 trapping sets (three types of trapping sets).

Step 4 Since there is only one trapping set in $\mathcal{A}$, there are no common VNs, and $\mathcal{Q}$ is an empty ordered set. Set $u = 1$.

Step 5 Now, $\mathcal{E}_1$ is the (5, 1)-type trapping set including 4 trapping sets, i.e., $\mathcal{J} = \mathcal{E}_1 = \{\{11,37,46,59,188\},\{20,77,93,139,162\},\{60,67,130,145,170\},\{64,75,170,179,190\}\}$. The appearance frequency of VN 170 is 2, and other VNs only appear once in these 4 trapping sets. We first insert VN 170 into $\mathcal{Q}$, i.e., $\mathcal{Q} = \{170\}$, and $\mathcal{J} = \{\{11, 37, 46, 59, 188\}, \{20, 77, 93, 139, 162\}\}$. Then, we insert VNs 11 and 20 into $\mathcal{Q}$. Now, $\mathcal{Q} = \{170, 11, 20\}$.

Step 6 We insert the first VN of {114, 154, 164, 199} in front of $\mathcal{Q}$ since there exists no common VN between this trapping set and $\mathcal{Q}$, i.e., $\mathcal{Q} = \{114, 170, 11, 20\}$.

Table 1
Dominant trapping sets of the (204, 102) Mackay code.

Type	Number	minimum d_c^2	average d_c^2	Type	Number	minimum d_c^2	average d_c^2
(4,0)	1	4.1849	4.1849	(5,3)	498	4.8739	16.0324
(5,1)	4	4.9746	7.6723	(7,3)	2248	4.8801	15.1087
(9,1)	34	6.8999	11.0467	(6,4)	7695	2.7132	18.3807
(7,1)	1	7.6917	7.6917	(8,4)	44656	3.6176	17.8413
(4,2)	21	3.9797	12.6847	(4,4)	1291	3.8193	21.3567
(6,2)	85	5.0367	12.0629	(7,5)	84574	3.0705	20.4221
(8,2)	357	5.5773	12.4480	(5,5)	9289	4.9746	22.7715
(9,3)	12021	4.0698	15.5103	(6,6)	68585	5.4932	23.8767
(3,3)	173	4.5091	21.9958	(8,6)	201173	5.5773	22.7468

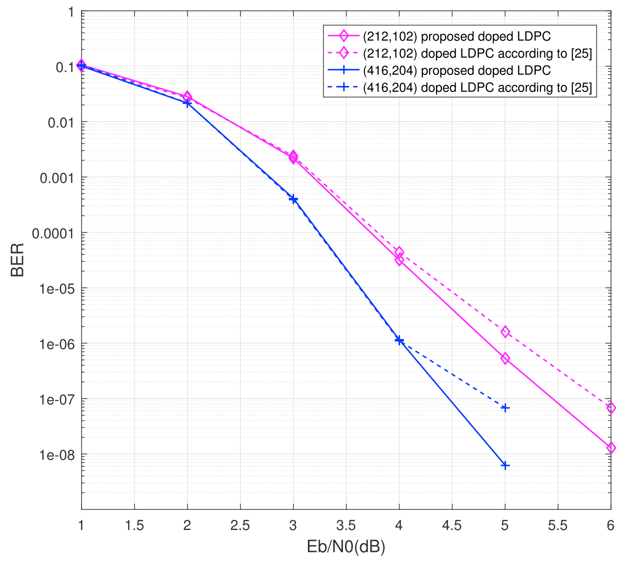


Fig. 5. Performance comparison of doped LDPC codes based on two different bit-selecting algorithms.

Step 7 Since $|\mathcal{Q}| < m$ (Here m is the message length of the component code), we obtain $u = 2$ and go to Step 5. The (9, 1)-type trapping set is considered next, and it has 34 trapping sets.

Step 8 We obtain $\mathcal{Q} = \{114,170,11,20,1,14,27,43,96,38,132,59,76,6,7,77\}$, and the intersection between $\mathcal{Q}$ and $\mathcal{A}$ is not empty.

Step 9 Algorithm 1 terminates since the cardinality of $\mathcal{Q}$ is larger than m (in the aftermentioned simulations, the (17, 9) QR code is used as the component code) now, and the VNs that will be used as the information bits of the component code are {114, 170, 11, 20, 1, 14, 27, 43, 96}.

Fig. 5 compares the error performance of doped LDPC codes in which the VNs of the component code are selected by our proposed bit-selecting algorithm and the bit-search algorithm in Ref. [27], respectively. The one-stage decoding algorithm is used. The (17, 9) QR code is used as the component code, and the (204, 102) Mackay code and the (408, 204) Mackay code are used as the original LDPC codes, respectively. As shown in the figure, our algorithm provides more than 0.3 dB gain at a Bit Error Rate (BER) of 10^{-7} for the two doped LDPC codes.

4. A two-stage decoding algorithm

In this section, a two-stage decoding algorithm is presented. In the first stage, the one-stage decoding algorithm mentioned previously is conducted. The second-stage decoding will start only if no valid codeword is output after the first stage. The main body of the second-stage decoding is still the log-BP algorithm, whereas the APP algorithm for decoding the component code will play a key role.

The component node (i.e., QR code) will transmit erroneous information continuously if the number of errors exceeds its error-correcting capacity. Hence, the confidence level of QR decoding needs to be evaluated. We define γ as the decoding confidence level of the component code after the first-stage decoding. The first stage of the two-stage decoding algorithm is conducted as follows:

Step 1 Initialization: Set $\gamma = 0$ and denote $\mathcal{C}_{qr}^{0}$ as the decoded QR codeword by making hard-decisions on the corresponding channel observations.

Step 2 Iterative decoding: Run the one-stage decoding λ times (a small modification is required since we need to determine the parameter γ). At the l-iteration, if $\mathcal{C}_{qr}^{l} = \mathcal{C}_{qr}^{l-1}$, $1 \le l \le \lambda$, then $\gamma = \gamma+1$; otherwise, γ is reset to 0. Here, $\mathcal{C}_{qr}^{l}$ represents the decoding decision corresponding to the QR code after l iterations.

Step 3 Decoding termination: If a valid codeword is an output or the preset maximum iteration number is reached in the iterative process of Step 2, terminate the decoding and record γ.

The second-stage decoding will start if a decoding failure is declared after the first stage. It is conducted in two modes.

1) **Mode-I:** When $\gamma > 0$, the QR code has been decoded successfully with a high probability. In this case, the information sent from the common VNs of the QR code and the LDPC code to the adjacent CNs of the LDPC code is set to the max/min LLR value depending on the decoded bit value. We denote the max/min LLR value by **MAXLLR**/− **MAXLLR** (say $\pm$ 30). Note that this 'initialization' is done only once.

For example, a VN transmits an LLR value "+30(−30)" to the neighboring CNs in the first iteration of the second-stage decoding if its corresponding hard-decision is a bit 0(1). The update rule for other VNs and CNs of the LDPC code keeps unchanged. (In this mode, the message update of the component code terminates since the QR decoding is successful with a high probability.)

2) **Mode-II:** When $\gamma = 0$, the confidence level of QR decoding is very low. In this case, we first determine the VN in the QR code with the least reliability. Then we set the initial LLR passing from this VN to its neighboring CNs as **MAXLLR** (or -**MAXLLR**) if the LLR output by the first-stage decoding is negative (positive). Subsequently, one starts the second-stage decoding, where the update rule for other VNs and CNs also keeps unchanged (i.e., the same update rule as the one-stage decoding algorithm). This method is similar to the process in the flipped BP algorithm, and we only flip the most unreliable VN and redo the iterative decoding.

Decoding failure may still occur even though two decoding modes are utilized in the second stage. It is because some VNs fail to escape from the trapping sets, and the algorithm does not converge to a valid codeword. Consequently, some further mandatory modifications are required. After the first-stage decoding, we record K VNs of the LDPC code not belonging to the component code and having the least reliabilities. Then we sort

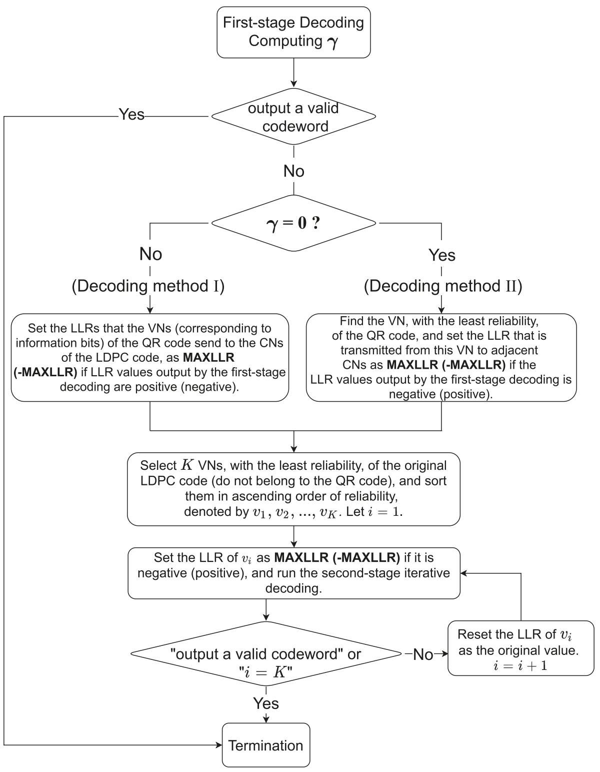

First-stage Decoding Computing γ
output a valid codeword
Yes
No
$\gamma = 0$?
No
Yes
(Decoding method I)
(Decoding method II)
Set the LLRs that the VNs (corresponding to information bits) of the QR code send to the CNs of the LDPC code, as **MAXLLR (-MAXLLR)** if LLR values output by the first-stage decoding are positive (negative).
Find the VN, with the least reliability, of the QR code, and set the LLR that is transmitted from this VN to adjacent CNs as **MAXLLR (-MAXLLR)** if the LLR values output by the first-stage decoding is negative (positive).
Select K VNs, with the least reliability, of the original LDPC code (do not belong to the QR code), and sort them in ascending order of reliability, denoted by $v_1, v_2, \ldots, v_K$. Let $i = 1$.
Set the LLR of v_i as **MAXLLR (-MAXLLR)** if it is negative (positive), and run the second-stage iterative decoding.
"output a valid codeword" or "$i = K$"
No
Reset the LLR of v_i as the original value. $i = i + 1$
Yes
Termination

Fig. 6. The flowchart of the proposed two-stage decoding algorithm.

them in an ascending order in terms of reliability (i.e., the magnitude of the channel observation) and denote them by v_1, v_2, …, v_K. Next, in the second-stage decoding, a K-round iterative decoding is conducted when a certain mode is determined. At the i-th round ($i = 1, 2, \ldots, K$), the i-th VN among the chosen K ones will send a specific LLR, which is set to **MAXLLR** (**-MAXLLR**) if v_i has a negative (positive) LLR after the first-stage decoding, to its neighboring CNs. If the decoder fails to converge, all LLRs are reset to the output LLRs of the first-stage decoding. Then i is incremented by 1, and the next decoding round begins. The detailed flowchart of the two-stage decoding algorithm is shown in Fig. 6.

5. Simulation results

In this section, we evaluate the performance of our proposed doped LDPC codes by taking two random LDPC and two structured LDPC codes as examples. Therein, dominant trapping sets are searched by the algorithm given in Ref. [28] and the trapping sets' influence on code performance is evaluated based on the method given in Ref. [29] with parameters $E_b/N_0 = 6.0$ dB, $l_{\max} = 3.6$, $l_{\min} = 1.0$ and $p = 10$ (these parameters are utilized to evaluate the trapping sets' influence on code performance, and we refer the readers to Ref. [29] for the definitions of $l_{\max}$, $l_{\min}$ and p). In order to maintain a low rate loss, we use the (17, 9, 5) and the (23, 12, 7) QR codes as the component codes. Unless specified otherwise, the maximum number of iterations is always set to 50 for the original LDPC and doped LDPC codes. When the two-stage decoding algorithm is utilized, the maximum number of iterations for the first and second stages are set to 10 and 40, respectively, with the parameter $K = 9$.

5.1. *Example 1*

Table 1 lists the types of trapping sets of the (204, 102) Mackay code. According to **Algorithm** 1, $\mathcal{A}$ is the (4, 0)-type trapping set while $\mathcal{E}$ consists of the (5,1)-type, the (9, 1)-type and the (7, 1)-type trapping sets. One (17, 9, 5) QR code is used as the component code of the doped LDPC code, and the index set of the 9 VNs selected by **Algorithm** 1 are {114, 170, 11, 20, 1, 14, 27, 43, 96}. The resultant doped LDPC code has a length of 212, and the code rate is reduced from 0.5 to 0.4811. As shown in Fig. 7, the doped LDPC code with the one-stage decoding algorithm performs approximately 1.6 dB better than the original LDPC code at a FER of 10^{-5}. However, another 0.3 dB gain can be obtained if the two-stage decoding algorithm is utilized.

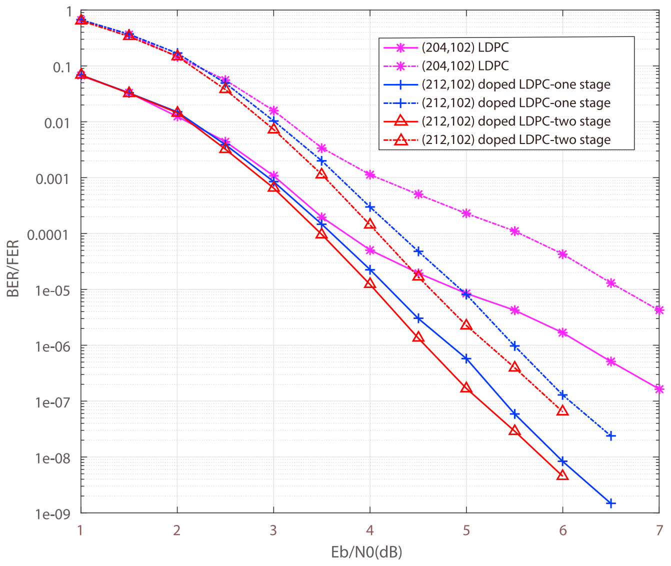


Fig. 7. The BER (solid lines) and FER (dot-dashed lines) performance comparison of the (204, 102) LDPC and the (212, 102) doped LDPC codes.

Table 2
Number of successful (N_{sd}) and unsuccessful (N_{usd}) decoding for $\gamma > 0$ and $\gamma = 0$ when 500000 doped LDPC codewords of length 212 are transmitted at each E_b/N_0.

E_b/N_0(dB)	$\gamma > 0$		$\gamma = 0$	
	N_{sd}	N_{usd}	N_{sd}	N_{usd}
1	327323	7377	33875	131425
1.5	392004	3748	25903	78345
2	444687	1545	16778	36990
2.5	476079	447	10250	13224
3	488064	104	8219	3613
3.5	487357	20	11811	812
4	475648	2	24198	152
4.5	451662	0	48303	35
5	419094	0	80901	5
5.5	389890	0	110108	2
6	376591	0	123409	0
6.5	382112	0	117888	0

In order to test the confidence level of QR decoding after the first-stage decoding, we record the number of codewords decoded successfully or unsuccessfully with $\gamma > 0$ or $\gamma = 0$ when 500000 codewords are transmitted for each E_b/N_0. As illustrated in Table 2, the confidence level of QR decoding is high for $\gamma > 0$ even at low E_b/N_0s. For example, when $E_b/N_0 = 1$ dB, among the 334700 cases ($= 377323 + 7377$) where $\gamma > 0$, 377323 cases show that the QR code is decoded successfully. The results confirm the rationality of distinguishing $\gamma > 0$ and $\gamma = 0$ in the second-stage decoding of the two-stage decoding algorithm.

5.2. *Example 2*

The information of the trapping sets for the (408, 204) Mackay code is given in Table 3. In this example, $\mathcal{A}$ is the (3, 1)-type trapping set while $\mathcal{E}$ consists of the (5, 1)-type, the (8, 2)-type and the (6, 2)-type trapping sets. The (17, 9, 5) QR code is used as the component code of the doped LDPC code, and the index set of the VNs chosen by **Algorithm** 1 is {122,

Table 3
Dominant trapping sets of the (408, 204) Mackay code.

Type	Number	minimum d_ϵ^2	average d_ϵ^2	Type	Number	minimum d_ϵ^2	average d_ϵ^2
(3,1)	2	4.0365	6.4223	(3,3)	156	11.4922	52.1651
(5,1)	1	8.8793	8.8793	(5,3)	195	12.1857	29.0657
(8,2)	42	5.5773	13.3553	(8,4)	5323	7.9594	24.4353
(6,2)	15	5.9696	13.4333	(4,4)	1210	8.2275	48.1440
(4,2)	12	7.4320	18.1337	(6,4)	2522	9.2436	30.3367
(2,2)	2	11.6083	13.0321	(9,5)	10831	8.9544	27.4922
(9,3)	1038	5.6787	18.5315	(7,5)	2642	12.2423	31.5660
(7,3)	506	6.4088	21.2139				

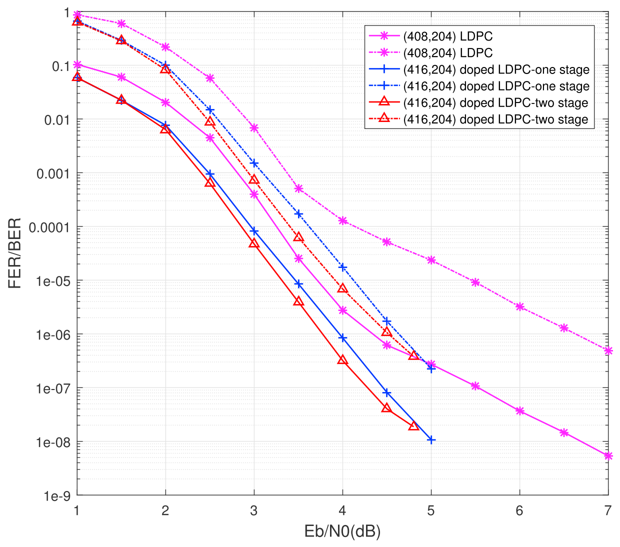


Fig. 8. The BER (solid lines) and FER (dot-dashed lines) performance comparisons of the (408, 204) LDPC and the (416, 204) doped LDPC codes.

36, 14, 53, 93, 52, 62, 71, 324}. The resultant doped LDPC code is of length 416 and rate 0.4904. As illustrated in Fig. 8, the proposed doped LDPC code significantly outperforms the original LDPC code with a slight rate loss. The gain provided by the doped LDPC code is approximately 2.0 dB at a FER of 2×10^{-6} compared with the original LDPC code. One also observes from the figure that the two-stage decoding algorithm provides a larger gain than the one-stage algorithm at the moderate-to-high E_b/N_0s.

5.3. Example 3

Table 4 lists the types of trapping sets for the (672, 336) Margulis code [33] and the minimum and average squared Euclidean distances. According to **Algorithm** 1, $\mathcal{A}$ is the (13, 3)-type trapping set while $\mathcal{E}$ consists of the (14, 4)-type, the (10, 4)-type and the (12, 4)-type trapping sets. The (23, 12, 7) QR code, i.e., Golay code, is used as the component code of the doped Margulis code, and 12 VNs with indices {8, 117, 270, 275, 317, 348, 439, 576, 1, 2, 31, 37} are chosen as the information section of the component code based on **Algorithm** 1. The resultant doped LDPC code is of length 683 and rate 0.4919. As illustrated in Fig. 9, the doped LDPC code with the one-stage decoding algorithm performs nearly the same as the original LDPC code in the whole E_b/N_0 regimes in terms of FER performance, whereas the former decoded by the two-stage decoding algorithm provides about 0.2 dB gain over the latter at a FER of 10^{-6}.

5.4. Example 4

Table 5 lists the types of trapping sets for the (208, 40) 5G LDPC code constructed based on the base graph II given in Ref. [34] and the minimum and average squared Euclidean distances. One observes that the values of "b" are evidently larger than those of the above three codes, which means thost trapping sets would have little effect on the decoding performance of this code. According to **Algorithm** 1, $\mathcal{A}$ is the (2, 7)-type trapping set while $\mathcal{E}$ consists of the (2, 11)-type, the (3, 11)-type and the (4, 11)-type trapping sets. The (7, 4, 3) Hamming code is used as the component code of the doped LDPC code, and 3 VNs with indices {19, 18, 17} are chosen as the information section of the component code based on **Algorithm** 1. The resultant doped LDPC code is of length 211 and rate 0.1896. As illustrated in Fig. 10, the doped LDPC code with the one-stage decoding algorithm performs almost as well as the original LDPC code, whereas the former decoded by the two-stage decoding algorithm is slightly better than the latter.

It is worth noting that the performance advantage of the proposed doped LDPC codes over the original LDPC codes shrinks significantly in the last two examples when compared with the first two. We believe that it is closely related to the characteristics of the trapping sets. As demonstrated in Tables 1 and 3, there exist (a, b)-types trapping sets where $b = 0$, 1 for the (204, 102) and the (408, 204) Mackay codes. Comparatively, the minimum values of the parameter 'b' are 3 and 7, respectively, for the (672, 336) Margulis code and the (208, 40) 5G LDPC code. As depicted in Section 2, the unsatisfied CNs (b represents the number of unsatisfied CNs in a (a, b)-type trapping set) help the trapped VNs escape from the trapping set. Therefore, the trapped VNs will receive sufficiently correct extrinsic messages from the unsatisfied CNs with a high probability when $b \geq 3$. Consequently, a more obvious effect on performance improvement is observed for those codes, with (a, 0)-type or (a, 1)-type trapping sets in our proposed doped LDPC framework.

Although the doped LDPC codes provide more coding gains than the

Table 4
Dominant trapping sets of the (672, 336) Margulis code.

Type	Number	minimum d_ϵ^2	average d_ϵ^2	Type	Number	minimum d_ϵ^2	average d_ϵ^2
(13,3)	1344	10.9128	20.1523	(11,5)	342048	10.9442	33.6160
(14,4)	107184	9.7602	25.7580	(7,5)	36288	18.9989	48.2877
(10,4)	10752	9.9493	28.3565	(5, 5)	12768	26.3916	72.5671
(12,4)	32928	10.0733	26.4102	(10,6)	931392	9.9493	41.5138
(8,4)	5040	13.1448	35.4228	(8,6)	348768	17.6409	49.3601
(6,4)	5040	27.3483	55.7902	(6,6)	91392	23.3436	63.4114
(4, 4)	2352	34.7501	85.1811	(13,7)	491904	12.9341	42.1621
(9,5)	125664	8.9544	39.2455	(11,7)	104832	19.2379	45.5565
(13,5)	885696	10.9128	30.9925	(9,7)	18816	24.4271	54.4713

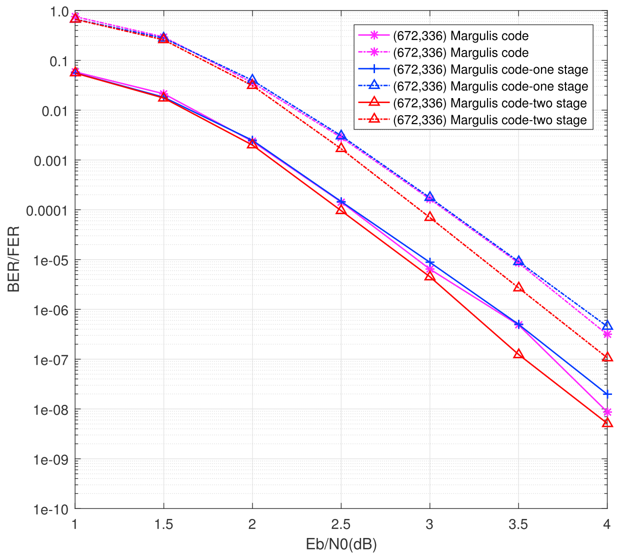


Fig. 9. The BER (solid lines) and FER (dot-dashed lines) performance comparison of the (672, 336) Margulis code and the (683, 336) doped LDPC code.

original LDPC codes, the transmission efficiency decreases accordingly. Given an (N, M) original LDPC code and an (n, m) component code, the code loss is $\frac{M}{N} - \frac{M}{N+n-m}$. For example, the (212, 102) doped LDPC code has an approximately 4% rate loss compared with its counterpart. However, the rate loss may be ignored for those long LDPC codes whose dominant trapping sets have relatively small parameters (say, the Reed-Solomon-based binary (2048, 1723) LDPC code included in the IEEE 802.3an standard. Its dominant trapping sets are reported to be the (8, 8) elementary trapping set [35]). Unfortunately, finding the dominant trapping sets of long LDPC codes is very time-consuming. In the future, we will develop an efficient algorithm to search the dominant trapping sets of long LDPC codes and apply the doped method to them.

6. Complexity analysis

The proposed doped LDPC codes include an additional QR component code compared to the original LDPC code, therefore possessing a higher decoding complexity at each iteration. The APP algorithm for decoding an (n, m) block code has a complexity of $O(2^{n-m}n)$ in which both real multiplications and real additions are included. In comparison, the log-BP algorithm for decoding an (N, M) LDPC code requires $O(Nw_c(w_c+w_r))$ real additions in each iteration where $w_c(w_r)$ denotes the column (row) weight.

Each iteration of the one-stage decoding algorithm consists of an iteration of the log-BP algorithm and the complete APP algorithm. Thus, we can roughly compare the complexity of the original LDPC and the doped LDPC codes by comparing the corresponding number of iterations. In Table 6, we list the average number of iterations for decoding the (204, 102) LDPC and the (212, 102) doped LDPC codes by the log-BP and the one-stage decoding algorithms, respectively, when 10000 all-zero codewords are transmitted at each E_b/N_0. As shown in this table, the average number of iterations for the doped LDPC code is close to that for the LDPC code.

Although the two-stage decoding algorithm seems more complex than the one-stage decoding counterpart, the second-stage decoding is only conducted when the first-stage decoding fails and the iterative decoding terminates once a valid codeword is found. As an illustration, we list the average number of rounds of the second-stage iterative decoding at each E_b/N_0 for the (212, 102) doped LDPC code in Table 7. Note that the average value is computed over 10000 tested codewords regardless of whether the second-stage decoding is conducted. (The second-stage decoding is not conducted if the first-stage decoding outputs a valid codeword.) As shown in the table, only 0.0006 average rounds of the second-stage iterative decoding are needed for each codeword at $E_b/N_0 = 5.0$ dB (It roughly corresponds to a 0.06% increase in complexity compared to the one-stage decoding algorithm.)

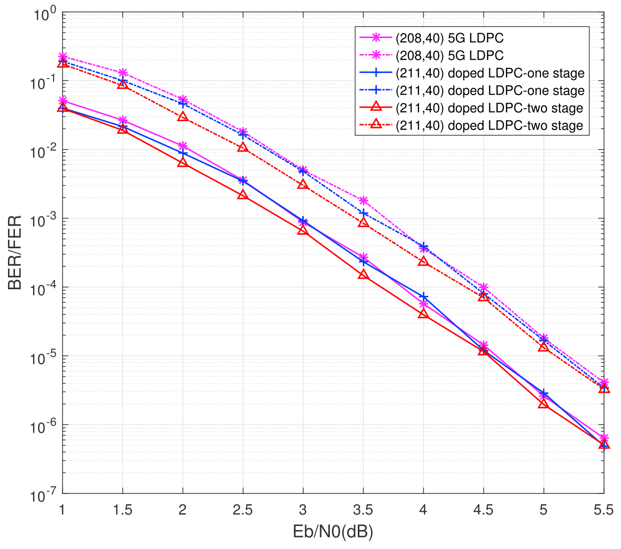


Fig. 10. The BER (solid lines) and FER (dot-dashed lines) performance comparison of the (208, 40) 5G LDPC and the (211, 40) doped LDPC codes.

Table 6
The comparison of average number of iterations for decoding the (204, 102) LDPC and the (212, 102) doped LDPC codes.

E_b/N_0 (dB)	average number of iterations for the (204, 102) LDPC code	average number of iterations for the (212, 102) doped LDPC code
1.0	35.90	35.55
2.0	14.39	14.02
3.0	4.84	4.82
4.0	2.55	2.64
5.0	1.71	1.79
6.0	1.24	1.28

Table 5
Dominant trapping sets of the (208, 40) 5G LDPC code.

Type	Number	minimum d_ϵ^2	average d_ϵ^2	Type	Number	minimum d_ϵ^2	average d_ϵ^2
(2,7)	4	0.0058	0.5711	(3,14)	28	0.0087	5.8866
(2,11)	4	0.0058	0.5711	(4,14)	22	0.0115	4.2179
(3,11)	4	0.0087	0.8567	(2,15)	16	0.0058	2.8401
(4,11)	4	0.0115	1.1422	(3,15)	20	0.0087	3.4532
(2, 12)	12	0.0058	3.7415	(4,15)	32	0.0115	7.0386
(3,12)	12	0.0087	5.6122	(4,16)	50	0.0007	4.7833
(2,13)	4	0.0058	0.5711	(2,16)	8	0.0058	4.1721
(3,13)	12	0.0087	5.6122	(3,16)	36	0.0087	4.7197
(4, 13)	4	0.0115	1.1422				

Table 7
average number of rounds of the second-stage decoding for decoding the (212, 102) doped LDPC codes.

E_b/N_0 (dB)	average rounds of the second-stage ecoding for all tested codewords
1.0	5.56
2.0	1.39
3.0	0.11
4.0	0.0048
5.0	0.00060

7. Conclusion

This work presents a novel design approach to constructing a class of doped LDPC codes, which integrate the Tanner graphs of an LDPC code and a short block code as a whole. In this framework, all the information bits of the short code are selected from the VNs of the LDPC code. They are selected based on the dominant trapping sets of LDPC codes. The construction mechanism is vastly different from the construction method of GLDPC codes, where a large number of SPC nodes are replaced with generalized constraint nodes, leading to a significant rate loss. Additionally, a specific two-stage decoding algorithm is developed. Simulation results show that the proposed doped LDPC codes can provide gains over the original LDPC codes with a slight rate loss and a moderate increase in computational complexity. Particularly, the proposed code design remarkably mitigates and even eliminates the error floor of original LDPC codes in some cases, using the proposed bit selecting algorithm and the two-stage decoding algorithm.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by China NSF under Grants No.61771081 and 62072064, the Fundamental Research Funds for the Central Universities(China) under Grant cstc2019jcyjmsxmX0110, the Project of Chongqing Natural Science Foundation under Grant CSTB2022NSCQ-MSX0990, Science and Technology Research Project of Chongqing Education Commission under Grant KJQN202000612, and the Venture and Innovation Support Program for Chongqing Overseas Returnees under Grant cx2020070.

References

[1] D.J.C. MacKay, R.M. Neal, Near Shannon limit performance of low density parity check codes, Electron, Letture 32 (1996) 1645–1646.
[2] 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 amendment 5: enhancements for higher throughput, in: IEEE Std 802.11n-2009, 2009, pp. 1–565.
[3] IEEE standard for air interface for broadband wireless access systems, in: IEEE Std 802.16-2017, 2018, pp. 1–2726.
[4] IEEE standard for information technology–telecommunications and information exchange between systems–local and metropolitan area networks–specific requirements part 3: carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications, in: IEEE Std 802.3-2008, 2008, pp. 1–2977.
[5] T. Richardson, S. Kudekar, Design of low-density parity check codes for 5G new radio, IEEE Commun. Mag. 56 (3) (2018) 28–34.
[6] C. Di, D. Proietti, E. Telatar, T. Richardson, R. Urbanke, Finite length analysis of low-density parity-check codes on the binary erasure channel, IEEE Trans. Inf. Theor. 48 (6) (2002) 1570–1579.
[7] T. Richardson, Error-floors of LDPC codes, in: Proc. 2003 41th Annual Allerton Conf. Communication, Control, and Computing(Allerton), IEEE, 2003, pp. 1426–1435.
[8] L. Dolecek, Z. Zhang, V. Anantharam, M.J. Wainwright, B. Nikolić, Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes, IEEE Trans. Inf. Theor. 56 (1) (2010) 181–201.
[9] R. Asvadi, A.H. Banihashemi, M. Ahmadian-Attari, Lowering the error floor of LDPC codes using cyclic liftings, IEEE Trans. Inf. Theor. 57 (4) (2011) 2213–2224.
[10] D.V. Nguyen, S.K. Chilappagari, M.W. Marcellin, B. Vasic, On the construction of structured LDPC codes free of small trapping sets, IEEE Trans. Inf. Theor. 58 (4) (2012) 2280–2302.
[11] J. Wang, L. Dolecek, R.D. Wesel, The cycle consistency matrix approach to absorbing sets in separable circulant-based LDPC codes, IEEE Trans. Inf. Theor. 59 (4) (2013) 2293–2314.
[12] X. Tao, Y. Li, Y. Liu, Z. Hu, On the construction of LDPC codes free of small trapping sets by controlling cycles, IEEE Commun. Lett. 22 (1) (2018) 9–12.
[13] A. Sarıduman, A.E. Pusane, Z.C. Taşkın, On the construction of regular QC-LDPC codes with low error floor, IEEE Commun. Lett. 24 (1) (2020) 25–28.
[14] B. Karimi, A.H. Banihashemi, Construction of QC-LDPC codes with low error floor by efficient systematic search and elimination of trapping sets, IEEE Trans. Commun. 68 (2) (2020) 697–712.
[15] M. Lentmaier, K. Zigangirov, On generalized low-density parity-check codes based on Hamming component codes, IEEE Commun. Lett. 3 (8) (1999) 248–250.
[16] G. Liva, W. Ryan, M. Chiani, Quasi-cyclic generalized LDPC codes with low error floors, IEEE Trans. Commun. 56 (1) (2008) 49–57.
[17] Y. Han, W.E. Ryan, Low-floor decoders for LDPC codes, IEEE Trans. Commun. 57 (6) (2009) 1663–1673.
[18] Z. Zhang, L. Dolecek, B. Nikolić, V. Anantharam, M.J. Wainwright, Design of LDPC decoders for improved low error rate performance: quantization and algorithm choices, IEEE Trans. Wireless Commun. 57 (11) (2009) 3258–3268.
[19] A.I.V. Casado, M. Griot, R.D. Wesel, LDPC decoders with informed dynamic scheduling, IEEE Trans. Commun. 58 (12) (2010) 3470–3479.
[20] F. Angarita, J. Valls, V. Almenar, V. Torres, Reduced-Complexity min-sum algorithm for decoding LDPC codes with low error-floor, IEEE Trans. Circuits Syst. I Regul Pap. 61 (7) (2014) 2150–2158.
[21] W. Li, J. Lin, Z. Wang, An efficient post processing scheme to lower the error floor of LDPC decoders, in: Proc. 2017 IEEE International Conference on Communication Technology(ICCT), IEEE, 2017, pp. 122–126.
[22] H.-C. Lee, P.-C. Chou, Y.-L. Ueng, An effective low-complexity error-floor lowering technique for high-rate QC-LDPC codes, IEEE Commun. Lett. 22 (10) (2018) 1988–1991.
[23] Y. Tao, S. Sun, Z. Zhang, Efficient post-processors for improving error-correcting performance of LDPC codes, IEEE Trans. Circuits Syst. I Regul Pap. 66 (10) (2019) 4032–4043.
[24] H. Hatami, D.G.M. Mitchell, D.J. Costello Jr., T.E. Fuja, A threshold-based min-sum algorithm to lower the error floors of quantized LDPC decoders, IEEE Trans. Commun. 68 (4) (2020) 2005–2015.
[25] T. Johansson, K. Zigangirov, A simple one-sweep algorithm for optimal APP symbol decoding of linear block codes, IEEE Trans. Inf. Theor. 44 (7) (1998) 3124–3129.
[26] Y. Li, X. Huang, C. Zhou, Z. Luo, H. Wu, A construction method of doped low-density parity-check codes based on quadratic residue codes, in: Proc. 2021 15th International Symposium on Medical Information and Communication Technology(ISMICT), IEEE, 2021, pp. 221–226.
[27] D. Yuan, L. Li, Y. Liu, Lowering the error floors of low-density parity-check codes with additional check nodes, in: Proc. 2017 IEEE Inf. Theory Workshop(ITW), IEEE, 2017, pp. 146–150.
[28] M. Karimi, A.H. Banihashemi, Efficient algorithm for finding dominant trapping sets of LDPC codes, IEEE Trans. Inf. Theor. 58 (11) (2012) 6942–6958.
[29] C. A. Cole, S. G. Wilson, E. K. Hall, and T. R. Giallorenzi, "A General Method for Finding Low Error Rates of LDPC Codes," arXiv preprint arXiv:0605051.
[30] O. Milenkovic, E. Soljanin, P. Whiting, Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles, IEEE Trans. Inf. Theor. 53 (1) (2007) 39–55.
[31] B. Frey, R. Koetter, A. Vardy, Signal-space characterization of iterative decoding, IEEE Trans. Inf. Theor. 47 (2) (2001) 766–781.
[32] X. Zhang, P.H. Siegel, Quantized iterative message passing decoders with low error floor for LDPC codes, IEEE Trans. Commun. 62 (1) (2014) 1–14.
[33] G.A. Margulis, Explicit constructions of graphs without short cycles and low density codes, Combinatorica 2 (1) (1982) 71–78.
[34] ETSI, 5G; NR; multiplexing and channel coding (Release 15), in: 3GPP TS 38.212 V15.2.0, 2018, pp. 1–101.
[35] C. Schlegel, S. Zhang, On the dynamics of the error floor behavior in (regular) LDPC codes, IEEE Trans. Inf. Theor. 56 (7) (2010) 3248–3264.