Efficient Search and Elimination of Harmful Objects for the Optimization of QC-SC-LDPC Codes

Massimo Battaglioni*, Franco Chiaraluce*, Marco Baldi*, and David G. M. Mitchell†
*Dipartimento di Ingegneria dell'Informazione, Università Politecnica delle Marche, Ancona, Italy
Email: {m.battaglioni, f.chiaraluce, m.baldi}@staff.univpm.it
†Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM 88011
Email: dgmm@nmsu.edu

***Abstract*—The error correction performance of low-density parity-check codes under iterative message-passing decoding is degraded by the presence of certain harmful objects existing in their Tanner graph representation. Depending on the context, such harmful objects are known as stopping sets, trapping sets, absorbing sets, or pseudocodewords. In this paper, we propose a general procedure, based on *edge spreading*, that enables the design of good quasi-cyclic spatially coupled low-density parity-check codes. These codes are derived from quasi-cyclic low-density parity-check (QC-LDPC) block codes and possess a significantly reduced multiplicity of harmful objects with respect to the original QC-LDPC block codes. The proposed procedure relies on a novel algorithm that greedily spans the search space of potential candidates to reduce the multiplicity of the target harmful objects. The effectiveness of the method is validated via examples and numerical computer simulations.**

***Index Terms*—Cycles, iterative decoding, LDPC convolutional codes, quasi-cyclic codes, spatially coupled codes, trapping sets.**

I. Introduction

Low-density parity-check (LDPC) block codes were first introduced by Gallager [1] and have attracted significant interest over time due to their capacity-approaching performance. The convolutional counterparts of LDPC block codes, called LDPC convolutional codes or spatially coupled LDPC codes (SC-LDPCCs), were first proposed in [2]. Further studies have shown that SC-LDPCCs are able to achieve the capacity of memoryless binary symmetric channels under iterative decoding based on belief propagation [3], [4].

It is well known that iterative algorithms used for decoding LDPC codes may get trapped in certain error patterns that arise due to structural imperfections in the code's Tanner graph. These objects may cause a severe degradation of the error correction performance, especially in the high signal-to-noise ratio region (i.e., the *error-floor* region). These harmful objects depend on the considered channel and the type of decoding algorithm in use. The concept of *stopping set* was introduced in [5], where the failures of iterative algorithms over the binary erasure channel are characterized. More complex channels, like the additive white Gaussian noise (AWGN) channel, require the definition of more subtle harmful objects. An early work in this direction is [6], where *trapping sets* are defined. A particularly harmful subclass of trapping sets, called *absorbing sets*, were shown to be stable under bit-flipping iterative decoders [7]. Many harmful trapping sets originate from *cycles* in the code's Tanner graph [8], [9].

SC-LDPCCs can be designed starting from LDPC block codes via an *edge spreading* procedure [10]. Clearly, the harmful objects of the SC-LDPCCs arise from related objects in the underlying LDPC block codes, and their multiplicity depends on the adopted edge spreading method. Some efforts have been devoted to the graph optimization from an absorbing set standpoint of array-based (AB) SC-LDPCCs [11]–[17]. These approaches have been restricted to certain code structures and harmful objects to enable a feasible search. Furthermore, most of these previous works have the limitation of excluding a priori many possible solutions of the problem in order to reduce the search space. Moreover, as shown in [13], [16], the multiplicity of harmful objects can be significantly reduced by increasing the memory of SC-LDPCCs. However, the computational complexity of previous approaches limits their viability to small memories. To the best of the authors' knowledge, a general scheme enabling the construction of optimized quasi-cyclic SC-LDPCCs (QC-SC-LDPCCs) (with respect to minimization of harmful objects) from QC-LDPC block codes with large memories is missing in the literature.

The objective of this paper is to propose an algorithm that, given any QC-LDPC block code, exploits a smart strategy to construct an optimized QC-SC-LDPCC by performing a greedy search over all candidates. This search attempts to minimize the multiplicity of the most harmful object (or combinations of objects) for the given channel and decoding algorithm. The effectiveness of the proposed procedure is demonstrated for several exemplary code constructions with variable code memories via enumeration of the target harmful objects and computer simulations.

The paper is organized as follows. In Section II, we introduce the notation used throughout the paper and some basic concepts of QC-LDPC block codes and SC-LDPCCs derived from them. In Section III, we focus on edge spreading matrices and the corresponding cycle properties. In Section IV, we describe the algorithm we propose. In Section V, we provide some code examples and assess their error rate performance. Finally, in Section VI, we draw some conclusions.

This material is based upon work supported by the National Science Foundation under Grant No. ECCS-1710920.

II. Definitions and notation

In this section we first introduce the notation for QC-LDPC codes and describe the edge spreading procedure to obtain QC-SC-LDPCCs from QC-LDPC block codes.

A. QC-LDPC codes

Let us consider a QC-LDPC block code, in which the parity-check matrix $\mathbf{H}$ is an $m \times n$ array of $N \times N$ circulant permutation matrices (CPMs) or all-zero matrices. We denote these matrices as $\mathbf{I}(p_{i,j})$, $0 \leq i \leq m-1$, $0 \leq j \leq n-1$, while N is the *lifting degree* of the code and $p_{i,j} \in \{-\infty, 0, 1, \ldots, N-1\}$. When $0 \leq p_{i,j} \leq N-1$, $\mathbf{I}(p_{i,j})$ is obtained from the identity matrix through a cyclic shift of its rows to the left by $p_{i,j}$ positions. Conventionally, we denote the all zero matrix by $\mathbf{I}(-\infty)$. The code length is $L = nN$. The *exponent matrix* of the code is the $m \times n$ matrix $\mathbf{P}$ having the values $p_{i,j}$ as its entries.

We associate a Tanner graph $\mathcal{G}(\mathbf{H})$ to any $\mathbf{H}$ in the usual way. The set of L variable nodes is denoted as $\mathcal{V}$ and the set of mN check nodes is denoted as $\mathcal{P}$. The set of edges is denoted as E. Thus, we can express $\mathcal{G}(\mathbf{H})$ as $\mathcal{G}(\mathcal{V} \cup \mathcal{P}, E)$. Let us consider the subgraph induced by a subset $\mathcal{D}$ of $\mathcal{V}$. We define $\mathcal{O}(\mathcal{D})$ as the set of neighboring check nodes with odd degree in such a subgraph. The *girth* of $\mathcal{G}(\mathbf{H})$, noted by g, is the length of the shortest cycle in the graph (or subgraph).

An (a, b) *absorbing set (AS)* is a subset $\mathcal{D}$ of $\mathcal{V}$ of size $a > 0$, with $\mathcal{O}(\mathcal{D})$ of size $b \geq 0$ and with the property that each variable node in $\mathcal{D}$ has strictly fewer neighbors in $\mathcal{O}(\mathcal{D})$ than in $\mathcal{P} \setminus \mathcal{O}(\mathcal{D})$. We say that an (a, b) AS $\mathcal{D}$ is an (a, b) *fully AS (FAS)* if, in addition, all nodes in $\mathcal{V} \setminus \mathcal{D}$ have strictly more neighbors in $\mathcal{P} \setminus \mathcal{O}(\mathcal{D})$ than in $\mathcal{O}(\mathcal{D})$.

For a QC-LDPC code, a necessary and sufficient condition for the existence of a cycle of length $2k$ in $\mathcal{G}(\mathbf{H})$ is [18]

$$\sum_{i=0}^{k-1} \left(p_{m_i,n_i} - p_{m_i,n_{i+1}}\right) = 0 \mod N, \tag{1}$$

where $n_k = n_0$, $m_i \neq m_{i+1}$, $n_i \neq n_{i+1}$. In the rest of the paper, with a slight abuse of notation, we refer to cycles in $\mathcal{G}(\mathbf{H})$ and cycles in $\mathbf{H}$ interchangeably.

B. QC-SC-LDPCCs based on QC-LDPC codes

The edge spreading procedure [13] is defined by an $m \times n$ (m_s+1)-ary *spreading matrix* $\mathbf{B}$, where m_s represents the *memory* of the resulting SC-LDPCC. The spreading matrix $\mathbf{B}$ can also be represented as an integer *spreading vector* $\mathbf{b}$ of length n, where the ith element, b_i, is obtained by replacing the ith column of $\mathbf{B}$, which is an (m_s+1)-ary vector of length m, to its decimal value. Conversely, $\mathbf{B}$ can be obtained from $\mathbf{b}$ by replacing each decimal entry with the associated (m_s+1)-ary column vector of length m. Given any column of $\mathbf{B}$, we assume, without loss of generality, that the most significant symbols are those with the smallest row indices. A straightforward example of conversion from $\mathbf{B}$ to $\mathbf{b}$ is reported in Example 1.

A *convolutional exponent matrix* derived from $\mathbf{B}$ has the following form

$$\mathbf{P}_{[0,\infty]} = \begin{bmatrix} \mathbf{P}_0 & & \\ \mathbf{P}_1 & \mathbf{P}_0 & \\ \vdots & \mathbf{P}_1 & \ddots \\ \mathbf{P}_{m_s} & \vdots & \ddots \\ & \mathbf{P}_{m_s} & \ddots \end{bmatrix},$$

where the (i,j)th entry of the $m \times n$ matrix $\mathbf{P}_k$, $k \in [0, 1, \ldots, m_s]$ is

$$P_k^{(i,j)} = \begin{cases} p_{i,j} & \text{if} \quad k = B_{i,j}, \\ -\infty & \text{if} \quad k \neq B_{i,j}, \end{cases}$$

and $B_{i,j}$ is the (i,j)th entry of $\mathbf{B}$. Let us remark that $-\infty$ represents void entries in the convolutional exponent matrix and corresponds to the $N \times N$ all-zero matrix in the related binary parity-check matrix. Notice that the entries of $\mathbf{P}_{[0,\infty]}$ which are off the main diagonal are $-\infty$ and have been omitted for the sake of readibility. The parity-check matrix of the QC-SC-LDPCC is then obtained as

$$\mathbf{H}_{[0,\infty]} = \begin{bmatrix} \mathbf{H}_0 & & \\ \mathbf{H}_1 & \mathbf{H}_0 & \\ \vdots & \mathbf{H}_1 & \ddots \\ \mathbf{H}_{m_s} & \vdots & \ddots \\ & \mathbf{H}_{m_s} & \ddots \end{bmatrix}, \tag{2}$$

where the appropriate $N \times N$ CPMs are substituted for the entries of $\mathbf{P}_{[0,\infty]}$ which have values in the set $\{0, 1, \ldots, N-1\}$, and the $N \times N$ all-zero matrix is substituted for the entries of $\mathbf{P}_{[0,\infty]}$ which are $-\infty$. In the subsequent sections, we will use also $\mathbf{H}_{[0,\mathcal{L}]}$ which represents a terminated version of $\mathbf{H}_{[0,\infty]}$, obtained by considering the first $(\mathcal{L}+m_s)Nm$ rows and $\mathcal{L}Nn$ columns of the semi-infinite parity-check matrix. Moreover, for the sake of readability, in the rest of the paper we refer to QC-SC-LDPCCs based on QC-LDPC codes as QC-SC codes.

Example 1 Consider the $(3, 5)$-regular array LDPC block code with the exponent matrix

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 2 & 4 & 1 & 3 \end{bmatrix} \tag{3}$$

and $N = 5$. Consider also the spreading matrix and the associated spreading vector, with $m_s = 2$,

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 & 3 & 6 & 21 & 10 \end{bmatrix}. \tag{4}$$

Then the constituent blocks of $\mathbf{P}$ are

$$\mathbf{P}_0 = \begin{bmatrix} 0 & 0 & 0 & - & - \\ 0 & - & - & - & 4 \\ - & 2 & 4 & 1 & - \end{bmatrix}, \mathbf{P}_1 = \begin{bmatrix} - & - & - & - & 0 \\ - & 1 & - & 3 & - \\ 0 & - & - & - & 3 \end{bmatrix},$$

$$\mathbf{P}_2 = \begin{bmatrix} - & - & - & 0 & - \\ - & - & 2 & - & - \\ - & - & - & - & - \end{bmatrix},$$

where, for simplicity, $-\infty$ has been expressed as $-$.

C. Exhaustive Search

According to the definition given in Section II-B, there are $(m_s+1)^{mn}$ possible spreading matrices. We use the following property from [19] to reduce the size of the search space, without loss of completeness.

Lemma 1 Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be exponent matrices. If $\mathbf{P}_1$ can be obtained by permuting the rows or the columns of $\mathbf{P}_2$, or it can be obtained by adding or subtracting (modulo N) the same constant to all the elements of a row or a column of $\mathbf{P}_2$, then the corresponding codes are *equivalent*.

It follows from Lemma 1 that the set of exponent matrices that contain at least one zero in each column represents, without loss of generality, the entire space of exponent matrices. Similarly, it is straightforward to show that the set of spreading matrices containing at least one zero in each column represents, without loss of generality, the entire space of spreading matrices. Each of the m entries of a column of $\mathbf{B}$ can assume values in $[0, 1, \ldots, m_s]$ and, thus, there are $(m_s + 1)^m$ possible columns. However, we can remove the m_s^m columns which do not contain any zero. It follows that

$$[(m_s + 1)^m - m_s^m]^n \tag{5}$$

spreading matrices cover the whole search space. It is straightforward to notice from (5) that the number of candidate edge spreading matrices becomes very large as the values of m, n, and m_s increase. For this reason, we propose, in Section IV, a novel procedure which allows one to distinguish "good" candidates from "bad" candidates. Such an algorithm, based on a *tree-search*, does not exclude, a priori, any candidate spreading matrix. Instead, "bad" candidates and their children are discarded by the algorithm during the search. In other words, the algorithm only keeps "good" candidates, under the assumption that the children of "bad" candidates are more likely to yield a higher multiplicity of harmful objects with respect to the children of "good" candidates. Numerical results provided in Section V confirm that the aforementioned assumption is reasonable, since the proposed algorithm outputs spreading matrices yielding a smaller multiplicity of harmful objects with respect to previous approaches.

III. Edge spreading matrices

As mentioned in Section II, trapping sets (and therefore ASs and FASs) originate from cycles, or clusters of cycles. In this section we prove conditions on the existence of cycles in $\mathbf{H}_{[0,\infty]}$; this allows us to derive the number of equations that must be checked for each candidate spreading matrix in order to verify if it is a "good" candidate or a "bad" candidate for the proposed algorithm. The "goodness" of a candidate is measured by the number of harmful objects of the underlying block code it can eliminate.

We say that a *block-cycle* of length λ exists in the Tanner graph corresponding to the parity-check matrix of the block code described by $\mathbf{P}$ if there exists an $m \times n$ submatrix of $\mathbf{P}$, denoted as $\mathbf{P}^\lambda$, containing λ of its non-void entries (and $-\infty$ elsewhere) such that (1) holds.

The *block-cycle distribution* (or *spectrum*) of $\mathbf{H}_{[0,\mathcal{L}]}$ is denoted as $\mathbf{D}^{\mathcal{L},\Lambda}$ and is a vector such that its ith entry $D_i^{\mathcal{L},\Lambda}$ represents the multiplicity of block-cycles of length $2i+4 \leq \Lambda$ in $\mathcal{G}(\mathbf{H}_{[0,\mathcal{L}]})$.

We calculate the *average number of block-cycles of length λ per node* E_λ as follows [11]:

1) evaluate the number of block-cycles spanning exactly i sections, $i \in [2, 3, \ldots, \lfloor \frac{\lambda}{4} \rfloor m_s + 1]$ as

$$K_i = D^{i,\lambda}_{\frac{\lambda-4}{2}} - \sum_{j=1}^{i-1} (i + 1 - j) K_j, \tag{6}$$

   where $K_1 = D^{1,\lambda}_{\frac{\lambda-4}{2}}$;
2) compute the average as

$$E_\lambda = \frac{\sum_{i=1}^{\lfloor \frac{\lambda}{4} \rfloor m_s + 1} K_i}{n}. \tag{7}$$

We also define $\mathbf{E}^\Lambda$ as the vector containing E_λ, $\forall \lambda \in [4, 6, \ldots, \Lambda]$, as its entries. A similar procedure can be used to compute the average number of (a, b) ASs, $E_{(a,b)}$. We introduce the concept of *harmful object* to encompass the cycles and ASs/fully ASs which deteriorate the performance of the considered codes in a single notion.

Remark 1 Consider a *block-cycle* of length λ, described by $\mathbf{P}^\lambda$, existing in the Tanner graph $\mathcal{G}(\mathbf{H})$ associated to the parity-check matrix of a block QC-LDPC code. Then, after the edge spreading procedure based on $\mathbf{B}$ is applied, $\mathbf{P}^\lambda$ also exists in $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ if and only if $\mathbf{B}^\lambda$, defined as

$$B^\lambda_{i,j} = \begin{cases} -\infty & \text{if} \quad P^\lambda_{i,j} = -\infty, \\ B_{i,j} & \text{otherwise}, \end{cases}$$

satisfies (1) over $\mathbb{Z}$.

Suppose now that the code defined by an exponent matrix $\mathbf{P}$ contains ν block-cycles. Given $\mathbf{B}$, we can extract all the submatrices $\mathbf{B}^{\lambda_i}$, $0 \leq i \leq \nu - 1$, that correspond to the block-cycles in the QC-LDPC code and check whether (1) is satisfied or not. If it is satisfied, then the block-cycle also exists in the QC-SC code; if it is not satisfied, then the block-cycle does not exist in the QC-SC code. In other words, given an exponent matrix and a spreading matrix, checking as many equations as the number of block-cycles in the exponent matrix will determine the number of block-cycles in $\mathbf{P}_{[0,\infty]}$. We also remark that a block-cycle in an exponent matrix corresponds to N cycles in the binary parity-check matrix.

Example 2 Consider the same code and the same spreading matrix as in Example 1 (see (3) and (4), respectively). $\mathcal{G}(\mathbf{H})$

contains twenty block-cycles of length $\lambda = 6$. For the sake of brevity, we only consider three of them, along with the corresponding entries of the spreading matrix

$$\mathbf{P}^{\lambda_0} = \begin{bmatrix} 0 & 0 & - & - & - \\ 0 & - & 2 & - & - \\ - & 2 & 4 & - & - \end{bmatrix} \mathbf{B}^{\lambda_0} = \begin{bmatrix} 0 & 0 & - & - & - \\ 0 & - & 2 & - & - \\ - & 0 & 0 & - & - \end{bmatrix},$$

$$\mathbf{P}^{\lambda_1} = \begin{bmatrix} - & 0 & 0 & - & - \\ 0 & - & 2 & - & - \\ 0 & 2 & - & - & - \end{bmatrix} \mathbf{B}^{\lambda_1} = \begin{bmatrix} - & 0 & 0 & - & - \\ 0 & - & 2 & - & - \\ 1 & 0 & - & - & - \end{bmatrix},$$

$$\mathbf{P}^{\lambda_2} = \begin{bmatrix} - & 0 & 0 & - & - \\ - & 1 & - & 3 & - \\ - & - & 4 & 1 & - \end{bmatrix} \mathbf{B}^{\lambda_2} = \begin{bmatrix} - & 0 & 0 & - & - \\ - & 1 & - & 1 & - \\ - & - & 0 & 0 & - \end{bmatrix}$$

Notice that $\mathbf{P}^{\lambda_i}$, $i = 0, 1, 2$, comply with (1), as they represent block-cycles in the array LDPC block code. Moreover, (1) is satisfied for $\mathbf{B}^{\lambda_2}$ but not for $\mathbf{B}^{\lambda_0}$ and $\mathbf{B}^{\lambda_1}$. In other words, $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ contains the block-cycles of length 6 corresponding to $\mathbf{P}^{\lambda_2}$, but not those associated to $\mathbf{P}^{\lambda_0}$ and $\mathbf{P}^{\lambda_1}$. The same procedure can be applied to test whether the remaining 17 block-cycles are also contained in $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ or not.

IV. A Greedy Algorithm to Construct Optimized QC-SC Codes

In this section we describe a general algorithm, named MInimization of HArmful Objects (MIHAO), which can be applied to an arbitrary harmful object (or objects) of interest to find a good QC-SC code. Given the exponent matrix of a QC-LDPC block code, we first determine which are the most harmful objects causing an error rate performance degradation. The pseudo-code describing the proposed recursive procedure is provided in Algorithm 1. We propose to use a tree-based search: the root node of the tree is the all-zero spreading matrix, which characterizes a QC-LDPC block code, while the lth tier contains all the spreading matrices with l non-zero entries that minimize the average number of harmful objects per node with respect to their parent node. If a parent node has no children nodes with better properties than its own, it is discarded, and the algorithm backtracks. If no specific stopping criterion is included, all candidates are tested; the node representing the spreading matrix (or vector) yielding the smallest average number of harmful objects per node is the output of the algorithm. Stopping criteria can be, for example, the maximum number of times the algorithm backtracks or the maximum number of tiers it spans.

We provide in the following a description of the functions used in Algorithm 1. The function edge_spread$(\mathbf{P}, \mathbf{B}, N)$ performs the edge spreading procedure as described in Section II-B, while count_elimin_objects$(\mathbf{P}, \mathbf{B})$ determines how many harmful objects are removed from $\mathbf{P}$ for a given $\mathbf{B}$. This is accomplished according to Remark 1, as shown in Example 2. The multiplicity of eliminated objects for each choice of $B_{i,j}$ is stored in a matrix denoted as $\mathbf{M}$. Then, the candidate base matrices are those maximizing the multiplicity of removed harmful objects. Finally, count_harmful_objects$(\mathbf{H}, \lambda)$ computes the average number of harmful objects of length λ per node in $\mathbf{H}$. This function uses the counting algorithm proposed in [20]. The metric we finally consider to determine whether the candidate is "good" or "bad" is the average number of harmful objects per node, as defined in (7). Note that the algorithm does not guarantee that the optimal solution, which is obviously unknown, will be the output but, as will be shown in Section V, it provides better solutions than the best ones available in the literature.

Algorithm 1

Input exponent matrix $\mathbf{P}$, circulant size N, size of harmful objects λ, all-zero spreading matrix $\mathbf{B}$, memory m_s

```
procedure MIHAO(P, N, λ, B, m_s)
    B_old ← B
    H ← edge_spread(P, B, N)
    C_old ← count_harmful_objects(H, λ)
    for i ← 0 to m do
        for j ← 0 to n do
            if B_{i,j} = 0 then
                for k ← 0 to m_s do
                    B_{i,j} ← k
                    M_{i,j}^(k) ← count_elimin_objects(P, B)
                    B_{i,j} ← 0
    M ← max_{0≤k≤m_s} M_{i,j}^(k)
    n_cands ← #(M_{i,j}^(k) = M)
    while !Stopping criterion do
        if n_cands > 0 then
            Randomly pick (i, j, k) such that M_{i,j}^(k) = M
            B_new ← B
            B_new^(i,j) ← k
            H ← edge_spread(P, B_new, N)
            C_new ← count_harmful_objects(H, λ)
            if C_new < C_old then
                B ← MIHAO(P, N, λ, B_new, m_s)
            else
                B ← B_old
            n_cands ← n_cands − 1
            M_{i,j}^(k) ← 0
        else
            B_out ← B_old
            return B_out
```

V. Numerical Results and Performance

We validate the procedure using array codes [21] and Tanner codes [22] as a benchmark, and we verify the expected performance improvement via Monte Carlo simulations.

A. Numerical results

It is known that the performance of $(3, n)$-regular array codes, which are characterized by $n = N = p$, where $p > 3$ is some prime number, is adversely affected by $(3, 3)$ ASs and $(4, 2)$ FASs. It can be easily shown that $(3, 3)$ ASs and $(4, 2)$ FASs derive from a cycle of length 6 and a cluster of two cycles of length 6, respectively [11]. We have applied

TABLE I
AVERAGE NUMBER OF $(3,3)$ ASS PER NODE $E_{(3,3)}$ IN AB SC-LDPC CODES WITH $m=3$, $m_s=1$

p	7	11	13	17	19	23
$E_{(3,3)}$	0.43	1	1.08	1.88	2.26	3.26
$E_{(3,3)}$ [11], [16]	0.43	1	1.23	1.88	2.68	3.78

Algorithm 1 to minimize their multiplicity in AB QC-SC codes when $m_s = 1$. The results are shown in Table I.

We have also considered the $(3,5)$-regular Tanner QC-LDPC code with $L = 155$, $g = 8$, code rate $R = \frac{2}{5}$ and

$$\hat{\mathbf{P}} = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 5 & 10 & 20 & 9 & 18 \\ 25 & 19 & 7 & 14 & 28 \end{bmatrix}. \tag{8}$$

The dominant trapping sets of this code are known to be $(8,2)$ ASs [23]. They consist of clusters of cycles of length 8, 10, 12, 14, and 16. The easiest approach to eliminate these sets is to target the shortest cycles for removal. By applying Algorithm 1 with the following inputs: $\hat{\mathbf{P}}$, $N = 31$, $\lambda = 8$, the all-zero spreading matrix $\mathbf{B}$, and $m_s = 1$, we obtain

$$\mathbf{b}_1 = \begin{bmatrix} 2 & 2 & 1 & 1 & 4 \end{bmatrix}, \tag{9}$$

which results in a QC-SC parity-check matrix with no cycles of length up to 8. We have $\mathbf{E}^{12} = \begin{bmatrix} 0 & 0 & 0 & 3.8 & 18.4 \end{bmatrix}$. One can also minimize the multiplicity of cycles of length 10 and 12, by applying Algorithm 1 with different values of λ. For example, the code with $g = 10$ we obtained, such that the multiplicity of cycles of length up to 12 is minimized, is

$$\mathbf{b}_2 = \begin{bmatrix} 2 & 1 & 6 & 1 & 5 \end{bmatrix}, \tag{10}$$

where $\mathbf{E}^{12} = \begin{bmatrix} 0 & 0 & 0 & 1.8 & 15 \end{bmatrix}$. Further improvement can be obtained by applying Algorithm 1 to eliminate all the block-cycles of length 10. This requires an increase in the memory to $m_s = 3$ and spreading vector

$$\mathbf{b}_3 = \begin{bmatrix} 35 & 12 & 50 & 50 & 15 \end{bmatrix}, \tag{11}$$

which yields $\mathbf{E}^{12} = \begin{bmatrix} 0 & 0 & 0 & 0 & 9.4 \end{bmatrix}$. Note that an exhaustive search for such a code demands a huge computational effort, since it would require to perform $69,343,957$ attempts.

Suppose we wish to reduce the multiplicity of cycles of length 12, which are known to combine into codewords of minimum weight 24. From the exponent matrix (8), Algorithm 1 with $m_s = 1$ outputs the spreading vector

$$\mathbf{b}_4 = \begin{bmatrix} 6 & 1 & 3 & 2 & 4 \end{bmatrix}. \tag{12}$$

In this case we have $\mathbf{E}^{12} = \begin{bmatrix} 0 & 0 & 0.6 & 3.2 & 14.2 \end{bmatrix}$.

As a final example, we consider the $(3,7)$-regular Tanner code with blocklength $L = 301$, $g = 8$, code rate $R = \frac{4}{7}$ and

$$\tilde{\mathbf{P}} = \begin{bmatrix} 1 & 4 & 16 & 21 & 41 & 35 & 11 \\ 6 & 24 & 10 & 40 & 31 & 38 & 23 \\ 36 & 15 & 17 & 25 & 14 & 13 & 9 \end{bmatrix}, \tag{13}$$

from which two QC-SC codes have been obtained with spreading vectors

TABLE II
AVERAGE SPEED UP OF ALGORITHM 1 WITH RESPECT TO RANDOM SEARCH

Code	$\mathbf{B}_1$	$\mathbf{B}_2$	$\mathbf{B}_3$	$\mathbf{B}_4$	$\mathbf{B}_6$
$\frac{t_{\text{ran}}}{t_{\text{alg}}}$	3.73	4.2	8.21	3.51	4.18

$$\mathbf{b}_5 = \begin{bmatrix} 3 & 4 & 2 & 4 & 1 & 6 & 6 \end{bmatrix}, \tag{14}$$

$$\mathbf{b}_6 = \begin{bmatrix} 5 & 3 & 1 & 4 & 6 & 2 & 4 \end{bmatrix}. \tag{15}$$

Vector $\mathbf{b}_5$ was randomly generated with $m_s = 1$, whereas $\mathbf{b}_6$ is the output of Algorithm 1 with inputs $\tilde{\mathbf{P}}$, $N = 43$, $\lambda = 12$, the all-zero spreading matrix $\mathbf{B}$, and $m_s = 1$. The respective block-cycle distributions of these two codes are

$$\mathbf{E}^{12} = \begin{bmatrix} 0 & 0 & 1.86 & 17.57 & 71.14 \end{bmatrix},$$

$$\mathbf{E}^{12} = \begin{bmatrix} 0 & 0 & 1.29 & 15.14 & 64 \end{bmatrix}.$$

We have compared the time taken by Algorithm 1 to return all these spreading matrices with the average time required to find spreading matrices with the same (or better) cycle spectra through random searches. The average speed up obtained is shown in Table II, where t_{ran} and t_{alg} are the times required by the random search and by Algorithm 1, respectively.

B. Error rate performance simulations

In this section we assess the performance of the newly designed codes described in Section V-A in terms of bit error rate (BER) via Monte Carlo simulations of binary phase shift keying (BPSK) modulated transmissions over the AWGN channel. We use a sliding window (SW) decoder with window size $W = 5(m_s + 1)$ performing 100 iterations.

First, we consider the $(3,13)$-regular array code and through simulations we assess the AB QC-SC code obtained by edge-spreading its exponent matrix $\mathbf{P}$ with the spreading matrix found by Algorithm 1 (the average number of $(3,3)$ ASs per node is given in Table I) and with a random spreading matrix. The results shown in Fig. 1 confirm that $(3,3)$ ASs have a significant impact on these codes and enforce the need of an effective design to reduce their multiplicity.

We also consider the $(3,5)$-regular Tanner code and assess the QC-SC codes obtained by edge-spreading (8) with $\mathbf{b}_1$ and $\mathbf{b}_2$. The results, shown in Fig. 2, confirm the effectiveness of Algorithm 1. If we analyze the decoding failure patterns of these codes we notice that, according to the analysis proposed in [24], many of them are caused by cycles of length 12. Moreover, for all cases of undetected error, the corresponding codewords can be decomposed into cycles of length 12. For this reason, we also assess the QC-SC code represented by $\mathbf{b}_4$. It can be noticed that, even though $\mathcal{G}(\mathbf{H}_{[0,\infty]})$ for (12) contains some block-cycles of length 8 and 10, there is an improvement due to the reduction of the multiplicity of block-cycles of length 12. The same approach has been followed for the QC-SC codes represented by $\mathbf{b}_5$ and $\mathbf{b}_6$ that are constructed from the $(3,7)$-regular Tanner code. According

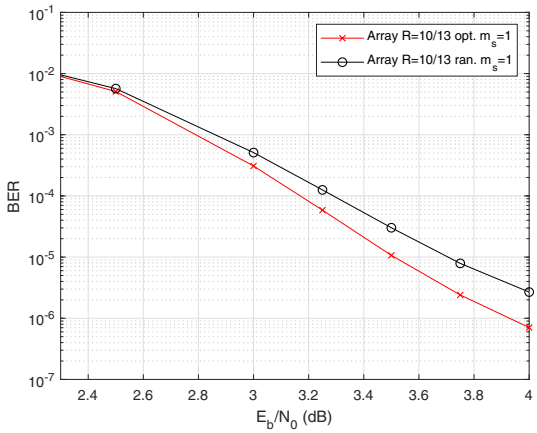

Fig. 1. Simulated performance of AB QC-SC codes as a function of the signal-to-noise ratio.

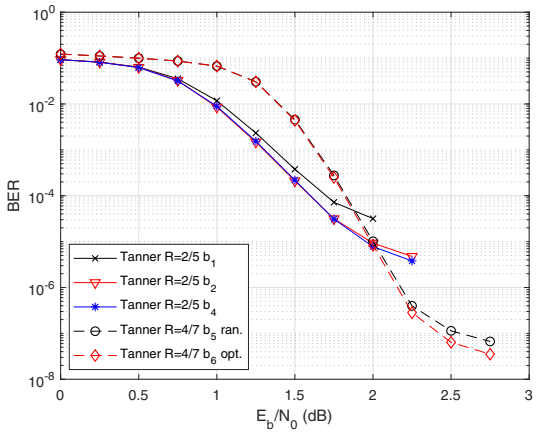

Fig. 2. Simulated performance of Tanner-based QC-SC codes as a function of the signal-to-noise ratio.

to their block-cycle spectra, the multiplicity of block-cycles of length 12 is minimized for (15). This is seen to have a positive impact on the BER performance in Fig. 2.

VI. Conclusion

We have proposed an efficient algorithm that enables the design of good QC-SC codes based on QC-LDPC block codes from the perspective of harmful objects. The algorithm is flexible and allows the analysis of codes with different structures and values of memory and rate. According to the proposed approach, many classes of harmful objects can be the target of a search-and-remove process aimed at optimizing codes in terms of their error rate performance.

References

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[2] A. Jiménez Felström and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2181–2191, Sep. 1999.

[3] M. Lentmaier, A. Sridharan, D. J. Costello, and K. S. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 5274–5289, Oct. 2010.

[4] S. Kudekar, T. J. Richardson, and R. L. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 803–834, Jan. 2011.

[5] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[6] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conf.*, Monticello, IL, Oct. 2003, pp. 1426–1435.

[7] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 181–201, Jan. 2010.

[8] Y. Hashemi and A. Banihashemi, "On characterization of elementary trapping sets of variable-regular LDPC codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 9, pp. 5188–5203, Sep. 2014.

[9] ——, "New characterization and efficient exhaustive search algorithm for leafless elementary trapping sets of variable-regular LDPC codes," *IEEE Trans. Inf. Theory*, vol. 62, no. 12, pp. 6713–6736, Dec. 2016.

[10] D. G. M. Mitchell, M. Lentmaier, and D. J. Costello, Jr., "Spatially coupled LDPC codes constructed from protographs," *IEEE Trans. Inf. Theory*, vol. 61, no. 9, pp. 4866–4889, Sep. 2015.

[11] D. G. M. Mitchell, L. Dolecek, and D. J. Costello, Jr., "Absorbing set characterization of array-based spatially coupled LDPC codes," in *Proc. IEEE ISIT 2014*, Honolulu, HI, USA, Jun. 2014, pp. 886–890.

[12] B. Amiri, A. Reisizadehmobarakeh, H. Esfahanizadeh, J. Kliewer, and L. Dolecek, "Optimized design of finite-length separable circulant-based spatially-coupled codes: An absorbing set-based analysis," *IEEE Trans. Commun.*, vol. 64, no. 3, pp. 918–931, Oct. 2016.

[13] D. G. M. Mitchell and E. Rosnes, "Edge spreading design of high rate array-based SC-LDPC codes," in *Proc. IEEE ISIT 2017*, Aachen, Germany, Jun. 2017, pp. 2940–2944.

[14] A. Beemer and C. A. Kelley, "Avoiding trapping sets in SC-LDPC codes under windowed decoding," in *Proc. ISITA 2016*, Monterey, CA, Oct. 2016, pp. 206–210.

[15] A. Beemer, S. Habib, C. A. Kelley, and J. Kliewer, "A generalized algebraic approach to optimizing SC-LDPC codes," in *Proc. 55th Annual Allerton Conf.*, Monticello, IL, Sep. 2017, pp. 672–679.

[16] H. Esfahanizadeh, A. Hareedy, and L. Dolecek, "A novel combinatorial framework to construct spatially-coupled codes: Minimum overlap partitioning," in *Proc. IEEE ISIT 2017*, Aachen, Germany, Jun. 2017, pp. 1693–1697.

[17] H. Esfahanizadeh, A. Hareedy, and L. Dolecek, "Finite-length construction of high performance spatially-coupled codes via optimized partitioning and lifting," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 3–16, Jan. 2019.

[18] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[19] M. Battaglioni, A. Tasdighi, G. Cancellieri, F. Chiaraluce, and M. Baldi, "Design and analysis of time-invariant SC-LDPC convolutional codes with small constraint length," *IEEE Trans. Commun.*, vol. 66, no. 3, pp. 918–931, Mar. 2018.

[20] H. Zhou and N. Goertz, "Cycle analysis of time-invariant LDPC convolutional codes," in *Proc. IEEE ICT 2010*, Doha, Qatar, Apr. 2010, pp. 23–28.

[21] J. L. Fan, "Array codes as low-density parity-check codes," in *Proc. 2nd Int. Symp. Turbo Codes*, Brest, France, Sep. 2000, pp. 543–546.

[22] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[23] S. Zhang and C. Schlegel, "Causes and dynamics of LDPC error floors on AWGN channels," in *Proc. 49th Annual Allerton Conf.*, Monticello, IL, Sep. 2011, pp. 1025–1032.

[24] M. Battaglioni, M. Baldi, and G. Cancellieri, "Connections between low-weight codewords and cycles in spatially coupled LDPC convolutional codes," *IEEE Trans. Commun.*, vol. 66, no. 8, pp. 3268–3280, Aug. 2018.