ISIT'2013 1569717663

A Revolving Iterative Algorithm for Decoding Algebraic Quasi-Cyclic LDPC Codes

Keke Liu, Shu Lin, Khaled Abdel-Ghaffar
Department of Electrical and Computer Engineering
University of California
Davis, CA, 95616, USA
Email: {kkeliu,shulin, ghaffar}@ucdavis.edu

***Abstract*—An effective reduced-complexity min-sum algorithm for decoding algebraic quasi-cyclic LDPC codes is presented. The proposed decoding algorithm significantly reduces the hardware implementation complexity, the size of memory required to store information, and the computational complexity of a decoder with no or a small loss in performance compared to the scaled min-sum algorithm.**

I. Introduction

Among various types of LDPC codes, the most preferred type in applications is the type of codes with quasi-cyclic (QC) structure. Many QC-LDPC codes have been chosen as the standard codes for various next generations of communication systems and they are appearing in recent data storage products. A QC-LDPC code is typically given by the null space of an *array of sparse circulants* of the same size over a finite field. Encoding of a QC-LDPC code can be efficiently implemented with simple shift-registers [1]. In hardware implementation of an iterative decoder, the QC structure simplifies the wire routing and allows partially parallel decoding which offers a trade-off between decoding complexity and decoding speed.

In this paper, we present an effective reduced-complexity min-sum (MS) algorithm for decoding QC-LDPC codes. This decoding algorithm is devised based on the quasi-cyclic structure and a *cyclic partition* of the parity-check matrix of a QC-LDPC code. The proposed decoding algorithm significantly reduces the hardware implementation complexity, the size of memory required to store information, and the computational complexity with no or a possibly small loss in performance compared to the scaled min-sum algorithm [1],[2]. This algorithm is particularly efficient for algebraic QC-LDPC codes constructed based on finite fields, combinatorial designs and finite geometries whose parity-check matrices have relatively high density and *large row redundancy* [1],[3]–[5].

The rest of this paper is organized as follows. Section II first presents a method to put the parity-check matrix of a QC-LDPC code into an array of submatrices of the same size with *block cyclic structure*. Based on this structure, a general algorithm is then introduced for decoding a QC-LDPC code using a small *subarray* of the entire parity-check matrix of the code in a *revolving manner*. Section III presents a revolving min-sum (RMS) algorithm for decoding QC-LDPC codes. Section IV analyzes the complexity of the RMS-decoder of a QC-LDPC code. Section V demonstrates the effectiveness of the proposed RMS-algorithm through two examples. Section VI concludes the paper with some remarks.

II. A General Revolving Algorithm for Decoding QC-LDPC Codes

Consider a QC-LDPC code $\mathcal{C}_{qc}$ given by the null space of a $J \times K$ array of $e \times e$ circulant matrices $\mathbf{H}_{qc}$. Label the rows and columns from 0 to $Je-1$ and 0 to $Ke-1$, respectively. Define the following two mappings: $\pi_{row} : i \rightarrow (i \bmod e)J + \lfloor i/e \rfloor$ for $0 \leq i < Je$ and $\pi_{col} : i \rightarrow (i \bmod e)K + \lfloor i/e \rfloor$ for $0 \leq i < Ke$. Then π_{row} and π_{col} define a permutation of the rows and a permutation of the columns of $\mathbf{H}_{qc}$, respectively. If we first permute the rows of $\mathbf{H}_{qc}$ based on π_{row} and then permute the columns of $\mathbf{H}_{qc}$ based on π_{col}, we obtain an $e \times e$ array $\mathbf{H}_{qc,perm}$ of $J \times K$ matrices as follows:

$$\mathbf{H}_{qc,perm} = \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \\ \vdots \\ \mathbf{H}_{e-1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_1 & \cdots & \mathbf{A}_{e-1} \\ \mathbf{A}_{e-1} & \mathbf{A}_0 & \cdots & \mathbf{A}_{e-2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_1 & \mathbf{A}_2 & \cdots & \mathbf{A}_0 \end{bmatrix}, \tag{1}$$

where $\mathbf{A}_i$, $0 \leq i < e$, is a $J \times K$ matrix over GF(2). The array $\mathbf{H}_{qc,perm}$ consists of e *row blocks* $\mathbf{H}_0, \mathbf{H}_1, \ldots, \mathbf{H}_{e-1}$. Each row block consists of a row of e $J \times K$ matrices and is a $J \times Ke$ sub-matrix of $\mathbf{H}_{qc,perm}$. From (1), we see that each row block is the *cyclic-shift* of the row block above it K positions to the right and the first row block is the cyclic-shift of the last row block K positions to the right. For $0 \leq i < e$, the sub-matrix $\mathbf{H}_i$ can be obtained from the first submatrix $\mathbf{H}_0$ by cyclically shifting all the rows of $\mathbf{H}_0$ iK positions to the right. The sub-matrices $\mathbf{H}_0, \mathbf{H}_1, \ldots, \mathbf{H}_{e-1}$ in (1) form a *cyclic partition* of the permuted matrix $\mathbf{H}_{qc,perm}$.

The null space of $\mathbf{H}_{qc,perm}$ also gives a QC-LDPC code, denoted by $\mathcal{C}_{qc,perm}$, in the sense that cyclically shifting each codeword K positions to the right is also a codeword [1],[6]. $\mathcal{C}_{qc,perm}$ is combinatorially equivalent to $\mathcal{C}_{qc}$ as it can be obtained by permuting the code symbols of each codeword in $\mathcal{C}_{qc}$ using the permutation π_{col}.

Let l be an integer where $1 \leq l \leq e$. For $i \geq 0$, form a

sequence of matrices $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$, each of size $Jl \times Ke$, where

$$\mathbf{H}_{qc,perm,i} = \left[\begin{array}{cccc} \mathbf{H}^T_{(il) \bmod e} & \mathbf{H}^T_{(il+1) \bmod e} & \cdots & \mathbf{H}^T_{(il+l-1) \bmod e} \end{array} \right]^T, \quad (2)$$

which is a submatrix of $\mathbf{H}_{qc,perm}$ and consists of l consecutive row blocks of $\mathbf{H}_{qc,perm}$ in cyclic order. From the construction, we readily see that any consecutive $\lceil e/l \rceil$ matrices in the sequence $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$ cover all the rows of $\mathbf{H}_{qc,perm}$. In the case for which l is a factor of e, any consecutive $c = e/l$ matrices in $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$ form a partition of $\mathbf{H}_{qc,perm}$. For $i \geq 0$, the matrix $\mathbf{H}_{qc,perm,i}$ can be obtained by cyclically shifting the rows of the first matrix $\mathbf{H}_{qc,perm,0}$ ilK positions to the right. This cyclic structure allows us to decode the code $\mathcal{C}_{qc,perm}$ based on the first matrix $\mathbf{H}_{qc,perm,0}$ of $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$ alone in a *revolving manner*.

The decoding of $\mathcal{C}_{qc,perm}$ based on its entire parity-check matrix $\mathbf{H}_{qc,perm}$ can be carried out in terms of the consecutive matrices in the sequence $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$. For $i \geq 0$, the i-th decoding iteration is defined as the information updating process based on the matrix $\mathbf{H}_{qc,perm,i}$ and passing the updated information to the next matrix $\mathbf{H}_{qc,perm,i+1}$ as input for decoding. Let I_{max} be the preset maximum number of iterations such that $I_{max} \geq \lceil e/l \rceil$, and label the decoding iterations from 0 to $I_{max}-1$. For $0 \leq i < I_{max}$, at the completion of the i-th decoding iteration based on $\mathbf{H}_{qc,perm,i}$, we pass the updated reliability information of the received symbols to $\mathbf{H}_{qc,perm,i+1}$ based on which we carry out the $(i+1)$-th iteration. At the end of each iteration, a hard-decision vector $\mathbf{z}$ is formed based on the reliabilities of the decoded symbols. Using this hard-decision vector $\mathbf{z}$, we compute the syndrome $\mathbf{s} = \mathbf{z}\mathbf{H}^T_{qc,perm}$. If $\mathbf{s} = \mathbf{0}$, $\mathbf{z}$ is a codeword in $\mathcal{C}_{qc,perm}$ and the decoding process stops, otherwise it continues until a preset number I_{max} of decoding iterations is reached. Since the sequence of matrices in $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$ is obtained by repeating the process of taking the top Jl consecutive rows from $\mathbf{H}_{qc,perm}$ and revolving (or cyclically shifting) $\mathbf{H}_{qc,perm}$ upwards by Jl rows, the above iterative decoding is referred to as a *revolving decoding* (RD) *algorithm*.

Consider the special case for which l is a factor of e. Let $c = e/l$. As pointed out earlier, the matrices $\mathbf{H}_{qc,perm,0}, \mathbf{H}_{qc,perm,1}, ..., \mathbf{H}_{qc,perm,c-1}$ form a cyclic partition of the entire parity-check matrix $\mathbf{H}_{qc,perm}$. In this case, the above RD-algorithm is similar to the layered decoding [7] or its variation, the staggered decoding in [8], in the sense that one decoding iteration based on a submatrix $\mathbf{H}_{qc,perm,i}, 0 \leq i < c$, in the RD-algorithm is similar to one sub-iteration (or step) based on one layer in the layered decoding algorithms proposed in [7] or [8]. In layered decoding, the parity-check matrix $\mathbf{H}_{qc}$ of a QC-LDPC code is in a form of an array of *circulant permutation matrices* (CPMs) and zero matrices (ZMs) of the same size. It contains a set of non-overlapped row blocks, say J of them, each consisting of a certain number of CPMs and a certain number of ZMs. These J row blocks correspond to J layers and form a partition of $\mathbf{H}_{qc}$. One decoding iteration based on the entire parity-check matrix $\mathbf{H}_{qc}$ consists of J sub-iterations corresponding to J layers. Each layer processes the received messages by the check nodes (CNs) of the Tanner graph of the code that correspond to one row block of CPMs and ZMs. Two subsets of CNs processed in two layers in one iteration are non-overlapping, and all the subsets of CNs, corresponding to all the layers in one iteration, form a partition of all the CNs in the Tanner graph of the code. The layered decoding of QC-LDPC codes is known for increasing decoding throughput and reducing the size of memory required for storing information.

It follows from the construction of the sequence $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$ of matrices that there may be overlapping between two matrices. For example, $\mathbf{H}_{qc,perm,i}$ and $\mathbf{H}_{qc,perm,i+\lfloor e/l \rfloor}$ may have common row blocks. Consequently, the RD-algorithm gives a more flexible decoder architecture for implementation than the layered decoding algorithm. The major advantage of the RD-algorithm over the layered decoding and its variations is that it can be carried out based on only the first submatrix $\mathbf{H}_{qc,perm,0}$ in the sequence $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$ alone (or any matrix in the sequence). This is due to the block cyclic structure of the matrices in $(\mathbf{H}_{qc,perm,i})_{i=0,1,...}$ such that each submatrix $\mathbf{H}_{qc,perm,i}$ can be obtained from $\mathbf{H}_{qc,perm,0}$ by cyclically shifting the rows of $\mathbf{H}_{qc,perm,0}$ ilK positions to the right. Since $\mathbf{H}_{qc,perm,0}$ is a submatrix of the parity-check matrix $\mathbf{H}_{qc,perm}$ of the QC-LDPC code $\mathcal{C}_{qc,perm}$ and can have a much smaller size and density than $\mathbf{H}_{qc,perm}$, using $\mathbf{H}_{qc,perm,0}$ *alone* for decoding with the RD-algorithm will significantly reduce the hardware complexity of the decoder in terms of the memory size for storing information, the number of CN processing units, and the number of wires connecting the CN processing units and VN (variable nodes) processing units.

The RD-algorithm based on $\mathbf{H}_{qc,perm,0}$ works as follows. The decoding process begins with initial symbol reliability information. For $i > 0$, at the completion of the $(i-1)$-th decoding iteration, the reliability vector of the decoded word is cyclically shifted to the left lK positions. This left shifted reliability vector together with the channel information are used as input to carry out the i-th decoding iteration based on $\mathbf{H}_{qc,perm,0}$. Note that using the left shifted reliability vector at the completion of the $(i-1)$-th decoding iteration as the input to the i-th decoding iteration based on $\mathbf{H}_{qc,perm,0}$ is equivalent to carrying out the i-th decoding iteration based on $\mathbf{H}_{qc,perm,i}$ using the (un-shifted) output reliability vector of the $(i-1)$-th decoding iteration as the symbol reliability information input. At the completion of each iteration, a hard-decision vector $\mathbf{z}$ is formed and its syndrome $\mathbf{s} = \mathbf{z}\mathbf{H}^T_{qc,perm}$ is computed. If $\mathbf{s} = \mathbf{0}$, $\mathbf{z}$ is a codeword in $\mathcal{C}_{qc,perm}$ and decoding stops, otherwise the revolving decoding process continues until either a codeword is found at the end of a certain iteration or the preset number I_{max} of iterations is reached.

III. A Revolving Min-Sum Algorithm

In each decoding iteration of the RD-algorithm, we could use either the sum-product algorithm (SPA) or the min-sum algorithm (MSA) [1] to update the reliabilities of the received

code symbols. Since our objective is to reduce the decoder complexity, we choose the MSA. This combined algorithm is referred to as a revolving MS (RMS)-algorithm.

Before we formulate the RMS-algorithm, we need to define some notations and expressions. Let $n = eK$ be the code length and $\mathbf{y} = (y_0, y_1, ..., y_{n-1})$ be the soft-decision received vector at the output of the receiver detector. We assume that the symbols of a codeword in $\mathcal{C}_{qc,perm}$ are transmitted over a binary AWGN channel with BPSK signaling. Let $\mathbf{H}_{qc,perm,0} = [h_{t,j}]_{0\le t<Jl, 0\le j<n}$. For $0 \le j < n$ and $0 \le t < Jl$, let $N(j) = \{t : 0 \le t < Jl, h_{t,j} = 1\}$ and $M(t) = \{j : 0 \le j < n, h_{t,j} = 1\}$, respectively. The notation (i) denotes the i-th iteration. Let I_{max} be the maximum number of iterations to be performed such that $I_{max} \ge \lceil e/l \rceil$. For $0 \le i \le I_{max}$, let $\mathbf{R}^{(i)} = (R_0^{(i)}, R_1^{(i)}, \ldots, R_{n-1}^{(i)})$ denote the reliability information vector at the beginning of the i-th iteration and let $\mathbf{F}^{(i)} = (F_0^{(i)}, F_1^{(i)}, \ldots, F_{n-1}^{(i)})$ denote the updated reliability information vector at the end of the i-th iteration. Let λ_{atten} denote the attenuation factor for MS-based reliability information updating. We label the iterations from 0 to $I_{max}-1$. Let f be a positive integer such that we include channel information for information update every f iterations.

With the above definitions and notations, the RMS-algorithm is formulated as follows:

RMS-Algorithm

Step 1. (**Initialization**) Set $i = 0$, and $R_j^{(0)} = y_j$ for $0 \le j < n$.

Step 2. (**Iteration**) Carry out the following i-th iteration:

(1) If $i \bmod f \ne 0$, compute the updated reliability information $\mathbf{F}^{(i)}$ based on $\mathbf{R}^{(i)}$ as follows: for $0 \le j < n$,

$$F_j^{(i)} = R_j^{(i)} + \lambda_{atten} \sum_{t\in N(j)} \left\{ \left[\prod_{j'\in M(t)\setminus j} \text{sign}\left(R_{j'}^{(i)}\right) \right] \times \left[\min\{|R_{j'}^{(i)}| : j' \in M(t)\setminus j\} \right] \right\}, \quad (3)$$

else, i.e., for $i \bmod f = 0$, compute the updated reliability information $\mathbf{F}^{(i)}$ based on $\mathbf{y}$ as follows: for $0 \le j < n$,

$$F_j^{(i)} = y_j + \lambda_{atten} \sum_{t\in N(j)} \left\{ \left[\prod_{j'\in M(t)\setminus j} \text{sign}\left(R_{j'}^{(i)}\right) \right] \times \left[\min\{|R_{j'}^{(i)}| : j' \in M(t)\setminus j\} \right] \right\}. \quad (4)$$

(2) Form the hard decision vector $\mathbf{z}$ based on $\mathbf{F}^{(i)}$ and compute its syndrome $\mathbf{s} = \mathbf{z}\cdot\mathbf{H}_{qc,perm}^T$. If $\mathbf{s} = \mathbf{0}$, go to Step 3. If $\mathbf{s} \ne \mathbf{0}$ go to Step 2-(3).

(3) If $i = I_{max} - 1$, stop decoding and declare a decoding failure. Otherwise, cyclically shift $\mathbf{F}^{(i)}$ by lK positions to the left to obtain $\mathbf{R}^{(i+1)}$, set $i \leftarrow i+1$ and return to Step 2-(1) to begin another iteration.

Step 3. Cyclically shift the hard decision vector $\mathbf{z}$ to the right by ilK positions to make it corresponding to the transmitted codeword and output the shifted $\mathbf{z}$ as the decoded codeword and stop decoding.

Different from the layered-decoding [7], the RMS-algorithm is performed based on a *single* sub-matrix $\mathbf{H}_{qc,perm,0}$ of the entire parity-check matrix $\mathbf{H}_{qc,perm}$, and requires that each VN pass the same message to all its adjacent CNs, which can further reduce the hardware complexity of VN processing units and the computational complexity. Also with the RMS-algorithm, all the memory for storing CN messages can be released at the completion of each iteration (corresponding to each layer in the layered-decoding), which further reduces the memory size. Moreover, in contrast to the staggered decoding in [8], the RMS-algorithm uses channel information for information updating every f iterations, which can further improve the error performance with properly chosen f.

IV. Analysis of Decoder Complexity

In this section we compare the decoder complexity for the RMS-, the MS- and the layered-MS-decoders. We assume maximum possible degree of parallel processing for each type of the decoders and suppose $\mathbf{H}_{qc,perm}$ has constant row weight ρ and column weight γ.

Since $\mathbf{H}_{qc,perm,0}$ contains only $l/e \times 100\%$ as many rows as $\mathbf{H}_{qc,perm}$, the number of CN processing units for the RMS-decoder based on $\mathbf{H}_{qc,perm,0}$ is $l/e \times 100\%$ of those of the MS- and the layered-MS-decoders implemented based on $\mathbf{H}_{qc,perm}$. Also, the number of wires required for the RMS-decoder based on $\mathbf{H}_{qc,perm,0}$ is $l/e \times 100\%$ of that of the MS-decoder implemented based on $\mathbf{H}_{qc,perm}$. Since in the RMS-decoder each VN passes the same message to all its adjacent CNs, the VN processing units in the RMS-decoder require less complexity for implementation than those in the MS-decoder and the layered-MS-decoder.

For memory size analysis, we assume that each real number (RN) message is quantized and represented by b bits for all types of decoders. The RMS-decoder needs nb binary memory units (BMUs) to store $\mathbf{y}$ received from the channel and nb BMUs to store $\mathbf{F}^{(i)}$. $\mathbf{R}^{(i)}$ can share memory units with $\mathbf{F}^{(i)}$. In addition, each CN in the Tanner graph of $\mathbf{H}_{qc,perm,0}$ requires $2(b-1)$ BMUs to store the two smallest magnitudes of the messages of its adjacent VNs, $\lceil \log_2 \rho \rceil$ BMUs to store the index of the smallest magnitude of the messages of its adjacent VNs, and ρ BMUs for the signs of the messages of its adjacent VNs. Since the Tanner graph of $\mathbf{H}_{qc,perm,0}$ has Jl CNs, the total number of BMUs required is $Jl[2(b-1) + \lceil \log_2 \rho \rceil + \rho] + 2nb$. Table I shows the memory size comparison among the RMS-, the layered-MS- and the MS-decoders.

For computational complexity comparison among the three types of decoders, we use the sorting algorithm in vol. 3, Sec. 5.3.3 of [9] to find the two smallest magnitudes. Let P be the least common multiple of l and e. Then it is fair to compare the computational complexity of P/e iterations of the MS-decoder and the layered-MS-decoder, and P/l iterations of the RMS-decoder. For the layered MS-decoder and the MS-decoder, each VN needs to subtract the previous CN message from its reliability value, which could result in more RN additions. Table I shows the number of RN additions required for P/l

TABLE I
COMPLEXITY COMPARISONS BETWEEN DIFFERENT DECODERS

	RMS	layered-MS	MS
Memory Size	$Jl[2(b-1)+\lceil\log_2\rho\rceil+\rho]+2nb$	$Je[2(b-1)+\lceil\log_2\rho\rceil+\rho]+nb$	$Je[2(b-1)+\lceil\log_2\rho\rceil+\rho]+(2b+\gamma)n$
# of RN additions	$JP(2\rho+\lceil\log_2\rho\rceil-2)$	$JP(3\rho+\lceil\log_2\rho\rceil-2)$	$JP(3\rho+\lceil\log_2\rho\rceil-2)$

iterations of the RMS-decoder and P/e iterations of the MS-decoder and the layered MS-decoder.

V. EXPERIMENTAL RESULTS

QC-LDPC codes can be constructed using different methods [1],[3]–[5]. The RMS-algorithm can be used to decode any of these codes to reduce decoding complexity. In the following, we use two algebraic QC-LDPC codes based on Latin squares [4] to demonstrate the effectiveness of the RMS-algorithm.

Example 1: The code considered in this example is the (992,750) QC-LDPC $\mathcal{C}_{qc}$ given in Example 1 of [4]. Its parity-check matrix $\mathbf{H}_{qc}$ is a 32×32 array of 31×31 CPMs and ZMs with the ZMs lying on the main diagonal of the array. It is a 992×992 matrix with both column and row weights equal to 31. The rank of this matrix is 242. Hence, it has 750 redundant rows (a very large row redundancy). The code $\mathcal{C}_{qc}$ has minimum distance at least 34. Permuting the rows and the columns $\mathbf{H}_{qc}$ based on permutations π_{row} and π_{col}, respectively, we obtain a 31×31 array $\mathbf{H}_{qc,perm}$ of permutation matrices of size 32×32. Each row block in $\mathbf{H}_{qc,perm}$ consists of 31 permutation matrices of size 32×32. It has block cyclic-shift structure. Each row block is a cyclic shift of the row block above it to the right by 32 positions and the first row block is the cyclic-shift of the last row block to the right by 32 positions. The QC-LDPC code $\mathcal{C}_{qc,perm}$ given by the null space of $\mathbf{H}_{qc,perm}$ is also a (992,750) QC-LDPC code which is combinatorially equivalent to the code $\mathcal{C}_{qc}$. Choose $l = 10$ and let $\mathbf{H}_{qc,perm,0}$ be the submatrix of $\mathbf{H}_{qc,perm}$ which consists of the top 10 row blocks of $\mathbf{H}_{qc,perm}$.

Suppose we decode the code based on $\mathbf{H}_{qc,perm,0}$ using the RMS-algorithm with 155 iterations, and based on the entire parity-check matrix $\mathbf{H}_{qc,perm}$ using the layered-MSA with 50 iterations and the MSA with 50 iterations. For $l = 10$, 155 iterations of the RMS-algorithm is equivalent to 50 iterations of the layered-MSA and the MSA based on the entire parity-check matrix $\mathbf{H}_{qc,perm}$. In order to optimize the performance, we choose $\lambda_{atten} = 0.25$ for the layered-MSA and the MSA, and $\lambda_{atten} = 0.375$ for the RMS-algorithm. For the RMS-algorithm, we choose $f = 3$. The bit error (BER) performances of this code decoded with the three algorithms are shown in Figure 1. We see that the performance curves of the three decoding algorithms basically overlap with each other. Also included in Figure 1 are the BER performances of the RMS-algorithm with 30 and 10 iterations (roughly 10 and 3 iterations of the layered-MSA and MSA based on $\mathbf{H}_{qc,perm}$, respectively). We see that the RMS decoding converges very fast. At the BER = 10^{-6}, the performance gap between 10 and 155 iterations is about 0.3 dB. Also included in Figure 1 is the performance of the code decoded based on $\mathbf{H}_{qc,perm}$ using the SPA with 50 iterations. We see that the performance gap between 155 iterations of the RMS-algorithm and 50 iterations of the SPA is about 0.1 dB.

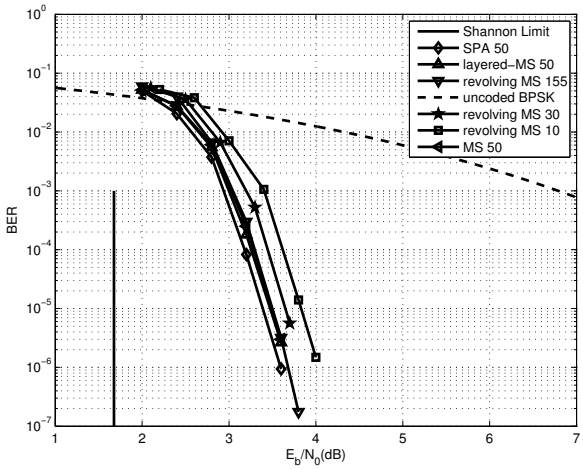

Fig. 1. The error performance of the (992,750) QC-LDPC code decoded with the RMS-, the MS-, the layered-MS-algorithms, and the SPA.

The number of wires and CN processing units required for the RMS-decoder is 32.26% of those required for the MS decoder. Based on simulations we find that at SNR=3.6 dB, the RMS-decoder requires only 36% as many RN additions per decoding process as the MS-decoder. If we choose 6-bit quantization for implementation, the RMS-decoder has an up to 89% memory saving compared to the MS-decoder. Compared with the layered-MS-decoder, the RMS-decoder needs the same number of RN additions per decoding process as the layered-MS-decoder at SNR=3.6 dB, and requires only 32.26% as many CN processing units as the layered-MS-decoder. With 6-bit quantization for implementation, the RMS-decoder has a 48.39% memory saving compared to the layered-MS-decoder. The RMS-decoder obviously reduces the decoding latency and enhances the decoder throughput compared to the layered-MS-decoder which processes only one row of CPMs/ZMs at a time. $\triangle\triangle$

Example 2: In this example, we consider the RMS decoding of the (4032,3304) QC-LDPC code $\mathcal{C}_{qc}$ given in Example 3 of [4], whose parity-check matrix $\mathbf{H}_{qc}$ is a 64×64 array $\mathbf{H}_{qc}$ of 63×63 CPMs and ZMs, with the ZMs lying on the main diagonal of the array. It is a 4032×4032 matrix with both column and row weights equal to 63. The rank of this matrix is 728. Hence, it has 3304 redundant rows (a very large row redundancy). The code $\mathcal{C}_{qc}$ has minimum distance at least 66 and its Tanner graph contains no harmful trapping set with size smaller than 60 [5].

Permuting the rows and columns of $\mathbf{H}_{qc}$ based on π_{row} and π_{col}, respectively, we obtain a 63×63 circular array $\mathbf{H}_{qc,perm}$ of 64×64 matrices of the form given by (1) which consists of 63 row blocks, each consisting of 63 permutation matrices of size 64×64. Each row-block is a 64×4032 matrix. The null space of $\mathbf{H}_{qc,perm}$ gives a QC-LDPC code $\mathcal{C}_{qc,perm}$ which is combinatorially equivalent to $\mathcal{C}_{qc}$. $\mathbf{H}_{qc,perm}$ has the block cyclic structure which allows us to decode it using the RMS-algorithm. Suppose we choose $l = 21$ which is a factor of $e = 63$. Then $\mathbf{H}_{qc,perm}$ can be partitioned into three submatrices $\mathbf{H}_{qc,perm,0}$, $\mathbf{H}_{qc,perm,1}$ and $\mathbf{H}_{qc,perm,2}$, each a 1344×4032 matrix. $\mathbf{H}_{qc,perm,1}$ and $\mathbf{H}_{qc,perm,2}$ can be obtained by cyclically shifting all the rows of $\mathbf{H}_{qc,perm,0}$ to the right 1344 and 2688 positions, respectively.

Suppose we decode the code based on the sub-matrix $\mathbf{H}_{qc,perm,0}$ using the RMS-algorithm with 150 iterations and based on the entire matrix $\mathbf{H}_{qc,perm}$ using the MSA with 50 iterations, respectively. We choose $\lambda_{atten} = 0.25$ for both the MSA and the RMS-algorithm, which provides good error performance for both algorithms. Note that three iterations of the RMS-algorithm are equivalent to one iteration of the MSA. For the RMS-algorithm, we choose $f = 3$. The BER performances of this code decoded using the MSA with 50 iterations, the SPA with 50 iterations and the RMS-algorithm with 15, 30, and 150 iterations are shown in Figure 2. We see that the performance curves of the code decoded using the RMS-algorithm with 150 iterations and the MSA with 50 iterations basically overlap with each other, and the RMS-algorithm performs only 0.12 dB away from the SPA algorithm with 50 iterations. Also we see that the performance curves of the code decoded with 15 and 30 iterations of the RMS-algorithm are very close to the performance curve of the code with 150 iterations (within 0.2 dB). So, the RMS decoding of the code converges very fast. This is attributed to the large row redundancy of the parity-check matrix of the code. If 6-bit quantization is used for decoding with the RMS-algorithm, from Figure 2, we see that there is no performance loss compared to the unquantized case.

Now we consider the extreme case for which $l = 1$. In this case, the RMS decoding of $\mathcal{C}_{qc,perm}$ is based on one row block of $\mathbf{H}_{qc,perm}$, i.e., $\mathbf{H}_{qc,perm}$ consists of a row of 63 permutaion matrices of size 64×64. This is equivalent to the typical layered decoding, except that all the 63 layers of decoding use the same submatrix $\mathbf{H}_{qc,perm,0}$. Typical layered decoding is implemented based on unpermuted array $\mathbf{H}_{qc}$, each layer corresponding to one row of CPMs/ZMs in $\mathbf{H}_{qc}$. We choose $f = 252$ for the RMS-decoder, and still assume 6-bit quantization and $\lambda_{atten} = 0.25$ for both the RMS-decoder and the layered-MS-decoder. Based on the analysis in Section IV, we find that the RMS-decoder can save 84.6% in memory compared to the layered-MS-decoder. Also, based on simulations we find that at SNR=4.0 dB, the RMS-decoder has 30% saving of RN additions per decoding process compared to the layered-MS-decoder. All the above advantages of the RMS-decoder over the layered-MS-decoder are gained at the expense of only about 0.1 dB performance loss as shown in

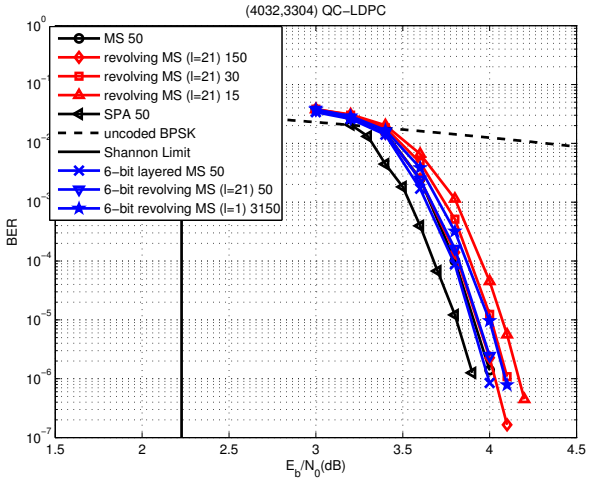

Fig. 2. The error performance of the (4032,3304) QC-LDPC code decoded with the RMS-, the layered-MS-, the MS-algorithm, and the SPA.

Figure 2. △△

VI. Conclusion and Remarks

In this paper, we presented a revolving-decoding algorithm for decoding QC-LDPC codes. The algorithm is devised based on the block cyclic structure and partitions of the parity-check matrices of QC-LDPC codes. This algorithm in conjunction with the min-sum algorithm results in not only a large reduction in hardware complexity and memory units but also in computational complexity with no or possibly only a small performance loss. Numerous simulation results, not reported here, show that the proposed RMS-algorithm is effective in decoding regular algebraic QC and cyclic LDPC codes but may not perform as well when decoding irregular codes.

References

[1] W. E. Ryan and S. Lin, *Channel Codes: Classical and Modern*. New York, NY: Cambridge Univ. Press, 2009.

[2] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X. -Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commu.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[3] L. Lan, L. Zeng, Y. Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar, Construction of quasi-cyclic LDPC codes for AWGN and binary erasure channels: A finite field approach, *IEEE Trans. Inf. Theory*, vol. 53, no. 7, pp. 2429–2458, Jul. 2007.

[4] L. Zhang, Q. Huang, S. Lin, K. Abdel-Ghaffar and I.F.Blake, "Quasi-cyclic LDPC codes: An algebraic construction, rank analysis, and codes on Latin squares," *IEEE Trans. Commun.*, vol. 58, no. 11, pp. 3126–3139, Nov. 2010.

[5] Q. Huang, Q. Diao, S. Lin, and K. Abdel-Ghaffar, "Cyclic and quasi-cyclic LDPC codes on constrained parity-check matrices and their trapping sets," *IEEE Trans. Inf. Theory*, vol. 58, no. 5, pp. 2648–2671, May 2012.

[6] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2004.

[7] D. E. Hocevar, "A reduced complexity decoder architecture via layered decoding of LDPC codes," *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, Austin, TX, Oct. 13–15, 2004, pp. 107–112.

[8] E. Yeo, P. Pakzad, B. Nikolic, V. Anantharam, "High-throughput low-density parity-check decoder architecture," *Proc. IEEE Globecom*, San Antonio, TX, Nov. 25–29, 2001, pp. 3019–3024.

[9] D. E. Knuth, *The Art of Computer Programming*, 2nd ed. New York: Addison-Wesley, 1998.