A Class of Doubly-Generalized LDPC Codes

Y. Min, F. C. M. Lau and C. K. Tse
Department of Electronic and Information Engineering, Hong Kong Polytechnic University, Hong Kong
Email: yue.min@connect.polyu.hk, encmlau@polyu.edu.hk, encktse@polyu.edu.hk

***Abstract*—We propose a class of doubly-generalized LDPC (DGLDPC) codes that use single-parity-check (SPC) codes as component codes at the super-variable nodes (SVNs) and SPC product-codes (SPC-PCs) as component codes at the super-check nodes (SCNs). We propose a low-complexity iterative decoding algorithm catered for the special structures of the SPC-PCs. Finally, we present the error performance and the convergence rate of the proposed DGLDPC codes.**

I. Introduction

Being one of the two known classes of Shannon limit-approaching codes, low-density parity-check (LDPC) codes beat their competitors (namely turbo codes) because of (i) their better error performance particularly at high code rate and (ii) the highly parallel decoder structure. The original LDPC codes are based on using repetition codes at the variable nodes (VNs) and using single-parity-check (SPC) codes at the check nodes (CNs) [1].

LDPC codes can also be generalized by replacing the SPC codes or/and the repetition codes with more complex linear block codes called constituent codes, and form the generalized LDPC (GLDPC) codes/doubly-generalized LDPC (DGLDPC) codes. Subsequently, the nodes become super-variable nodes (SVNs) and super-check nodes (SCNs). Many kinds of short-length component codes have been considered for use in GLDPC/DGLDPC codes. They include Hamming component codes [2]–[4]; Bose-Chaudhuri-Hocquenghem (BCH) component codes [3], [5]; Reed-Solomon (RS) component codes [5]; Reed-Muller component codes [6]; and hybrid component codes [7], [8]. Comparing with the LDPC codes, GLDPC codes and DGLDPC codes possess several advantages including better error performance, faster convergence rate and good performance at low rates. However, GLDPC/DGLDPC decoder complexity is much higher since more complicated constraints are introduced. Hence, finding proper constituent codes that can achieve a given error performance at a tolerable computation complexity is a key challenge in designing good GLDPC/DGLDPC codes.

In [9], it has been demonstrated that product codes can force the probability of bit error to zero when the number of dimensions goes to infinite. In [10], it has further been shown that SPC-PCs with check on checks can achieve error-free transmission within 2 dB of the theoretical capacity by using an iterative decoding algorithm. The SPC-PCs have surprisingly good performance even for high code rate and relatively short block length because of their outstanding *minimum distance* property. It can achieve an asymptotic coding gain of 11.3 dB [11]. Besides, SPC-PCs are suitable for effective decoding algorithm, in which the *extrinsic* information of each bit is exchanged between different dimensions of the product codes for a fixed number of iterations. Because of the above advantages, we propose using SPC-PCs component codes at SCNs in GLDPC/DGLDPC codes. We further propose applying SPCs at SVNs because SPCs are simple to decode and can raise the overall rate of the DGLDPC codes.

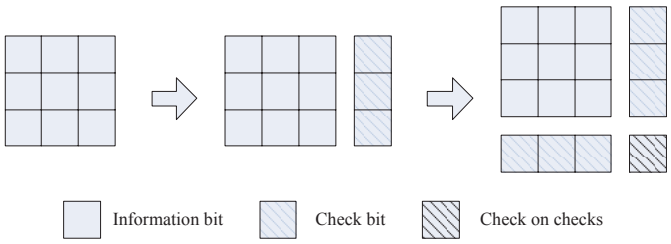


Fig. 1: The structure of the $(4,3)^2$ single-parity-check product-code (SPC-PC). 9 information bits are arranged in a 2-dimensional 3×3 structure. Check bits are added in the first dimension and then in the second dimension.

We organize this paper as follows. In Section II, we give a review of multi-dimensional SPC-PCs in terms of their structure and their *minimum distance* property. In Section III, we describe an iterative algorithm for decoding SPC-PCs. Compared with the maximum *a posteriori* probability (MAP) decoder, the iterative algorithm has a lower complexity. Finally, in Section IV, we present our proposed DGLDPC codes, in which SPC-PCs are used as component codes at SCNs and SPCs are used at SVNs. We present the error performance and the convergence rate of the proposed DGLDPC codes.

II. Single-Parity-Check Product-Codes

SPC-PCs, denoted by $(n_{spc}, n_{spc}-1)^D$, is referred to as a kind of D-dimensional product code concatenated with $(n_{spc}, n_{spc}-1)$ SPC component codes in each dimension. A $(n_{spc}, n_{spc}-1)^D$ SPC-PC can be constructed as follows. Firstly, the information bits are arranged in a D-dimensional hypercube. Secondly, each $n_{spc}-1$ information bits are encoded by a $(n_{spc}, n_{spc}-1)$ SPC in each dimension [12]. Figure 1 shows the process of constructing a $(4,3)^2$ SPC-PC. In particular, 9 information bits are arranged as a 2-dimension array (size 3×3), and then encoded by the $(4,3)$ SPC code in a row-wise and column-wise manner. Finally, a check-on-checks bit is added.

For a $(n_{spc}, n_{spc}-1)^D$ SPC-PC, in general, the number of

information bits is equal to

$$k_{spc-pc} = (n_{spc} - 1)^D, \tag{1}$$

the code length is

$$n_{spc-pc} = n_{spc}^D, \tag{2}$$

the number of valid check bits is

$$m_{spc-pc} = n_{spc}^D - (n_{spc} - 1)^D, \tag{3}$$

and hence the code rate is equal to

$$r_{spc-pc} = \frac{k}{n} = \frac{(n_{spc} - 1)^D}{n_{spc}^D}. \tag{4}$$

The corresponding parity-check matrix is of size $m_{spc-pc} \times n_{spc-pc}$.

The *minimum distance* of the $(n_{spc}, n_{spc}-1)^D$ SPC-PC has been shown equal to [12]

$$d_{min} = 2^D, \tag{5}$$

and the number of codewords with this *minimum distance* equals

$$\mathcal{N}_{d_{min}} = \left(\frac{n_{spc}(n_{spc} - 1)}{2}\right)^D. \tag{6}$$

It has also been proved that DGLDPC codes with constituent codes having a *minimum distance* $d_{min} \geq 3$ performs well asymptotically [13]. Since the *minimum distance* of the $(n_{spc}, n_{spc} - 1)^D$ SPC-PC equals $d_{min} = 2^D$ [12], such an outstanding distance property makes SPC-PC an attractive candidate for the constituent code of DGLDPC codes.

III. Decoding Algorithm

The belief propagation (BP) algorithm can be used to decode DGLDPC codes. We consider the *extrinsic* information exchange between the super-variable nodes (SVNs) and the super-check nodes (SCNs), based on the bipartite graph of the adjacency matrix, as the role of a *global* decoder. Besides, each super node is regarded as a *local* decoder. A soft-input soft-output (SISO) decoding algorithm, e.g., maximum *a posteriori* probability (MAP) algorithm, is employed in the *local* decoder to generate the *extrinsic* information for the *global* decoder.

Since the complexity of the DGLDPC decoder is dominated by the complexity of the *local* decoders at the SCNs [14] [15], we focus on reducing the complexity of such decoders. At each SCN, we decode each of the SPC component codes of the SPC-PC with an independent SISO decoder based on the BP algorithm [16]. Then, the *extrinsic* information of each SPC decoder output can be exchanged among the different dimensions of the SPC-PC by using a small number of *local* turbo iterations. By adjusting the number of *local* turbo iterations, a tradeoff between error performance and computation complexity can be obtained.

Suppose the $(n_{spc}, n_{spc} - 1)^D$ SPC-PC is used as the constituent code in each SCN. Each $(n_{spc}, n_{spc} - 1)$ component SPC code in each dimension is considered as an independent code and uses an independent decoder. The iterative decoding algorithm of the SPC-PC decoder used at the SCN is summarized as follows. Unless otherwise stated, all messages are in log-likelihood-ratio (LLR) format.

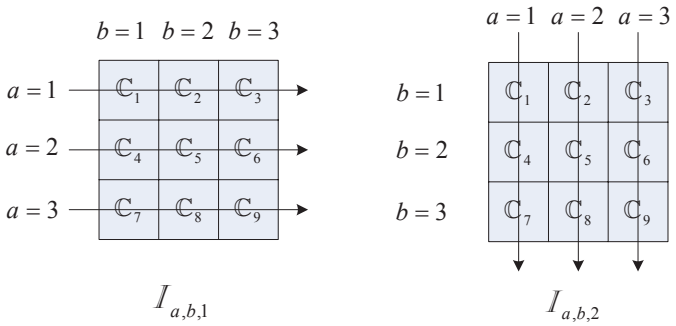


Fig. 2: Illustration of notations with a $(3, 2)^2$ SPC-PC.

A. Initialization

a) Initialize the channel messages: Each SCN treats the incoming messages passed from the SVNs as *channel messages*. For a $(n_{spc}, n_{spc} - 1)^D$ SPC-PC used at the SCN, there are $(n_{spc})^D$ such *channel messages* denoted by $\mathbb{C}_t$ $(t = 1, 2, ..., (n_{spc})^D)$. Also denote $\mathbb{I}$ as the set of messages. For a given dimension $c = 1, 2, ..., D$, there are $(n_{spc})^{D-1}$ SPC codes; and for each of these SPC codes, there are n_{spc} messages. Then, denote $\mathbb{I}_{a,b,c}$ as the *channel message* corresponding to the b-th bit of the a-th SPC component code in the c-th dimension of the product code ($a = 1, 2, ..., (n_{spc})^{D-1}$; $b = 1, 2, ..., n_{spc}$ and $c = 1, 2, ..., D$).

Figure 2 illustrates the above notations based on a $(3, 2)^2$ SPC-PC i.e., a 2-dimensional SPC-PC with 9 coded bits. There are 9 *channel messages* denoted by $\mathbb{C}_t$ $(t = 1, 2, ..., 9)$. The messages can be viewed from 2 different dimensions ($c = 1, 2$). For a given dimension, there are 3 SPCs; and for each SPC, there are 3 messages.

b) Initialize the extrinsic LLR values:: Denote the *extrinsic* LLR value corresponding to the b-th bit of the a-th SPC component code in the c-th dimension of the SPC-PC as $\mathbb{E}_{a,b,c}$ and initialize it to zero, i.e.,

$$\mathbb{E}_{a,b,c} = 0, \tag{7}$$

where $a = 1, 2, ..., (n_{spc})^{D-1}$, $b = 1, 2, ..., n_{spc}$ and $c = 1, 2, ..., D$.

B. Iteration

Denote $\mathbb{A}_{a,b,c}$ as the *a priori* LLR value corresponding to the b-th bit of the a-th SPC component code in the c-th dimension of the SPC-PC.

Step 1: Set the *local* iteration number $\tau = 1$.

Step 2: Set the dimension index $c = 1$.

Step 3: Compute the *a priori* LLR value for each bit in each SPC component code in the c-th dimension by summing the *extrinsic* LLR values of the corresponding bit in other dimensions of the SPC-PC, i.e.,

$$\mathbb{A}_{a,b,c} = \sum\nolimits_{\ell=1,\ell\neq c}^{D} \mathbb{E}_{a,b,\ell} \tag{8}$$

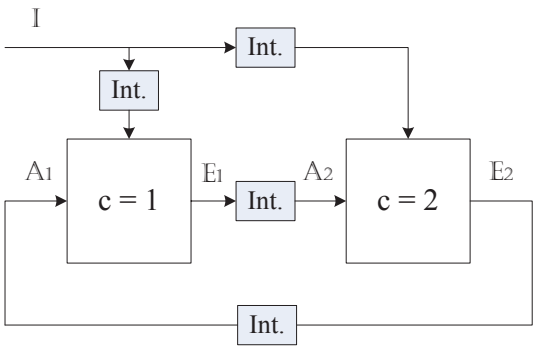


Fig. 3: Passing of the *extrinsic* LLR values in a 2-dimensional SPC-PC decoder. $\mathbb{I}$ denotes the *channel messages* in the form of LLR, $\mathbb{A}$ denotes the *a priori* LLR values at the input of a SISO decoder and $\mathbb{E}$ denotes the *extrinsic* LLR values at the output of a SISO decoder. The subscript corresponds to the index of dimension of the SISO decoder. "Int." denotes an interleaver.

where $a = 1, 2, ..., (n_{spc})^{D-1}$ and $b = 1, 2, ..., n_{spc}$.

Step 4: Compute the *extrinsic* LLR value for each of the n_{spc} bits in each of the $(n_{spc})^{D-1}$ SPC component codes using

$$\mathbb{E}_{a,b,c} = 2\tanh^{-1}\left(\prod_{\ell=1,\ell\neq b}^{n_{spc}} \tanh\left(\frac{\mathbb{I}_{a,\ell,c}+\mathbb{A}_{a,\ell,c}}{2}\right)\right) \quad (9)$$

$$= \log\left(\frac{1+\prod_{\ell=1,\ell\neq b}^{n_{spc}} \tanh\left(\frac{\mathbb{I}_{a,\ell,c}+\mathbb{A}_{a,\ell,c}}{2}\right)}{1-\prod_{\ell=1,\ell\neq b}^{n_{spc}} \tanh\left(\frac{\mathbb{I}_{a,\ell,c}+\mathbb{A}_{a,\ell,c}}{2}\right)}\right), \quad (10)$$

where $a = 1, 2, ..., (n_{spc})^{D-1}$ and $b = 1, 2, ..., n_{spc}$. Note that these *extrinsic* LLR values are used for updating the *a priori* LLR values for the bits in other dimensions.

Step 5: The computation for the c-th dimension is completed. Increment the value of the dimension index c by 1, i.e., set $c = c + 1$. If $c \leq D$, go to Step 3.

Step 6: One *local* turbo iteration is completed. Increment the *local* iteration number τ by 1, i.e., set $\tau = \tau + 1$. If τ does not exceed a pre-set maximum *local* iteration number τ_{max}, go to Step 2.

Step 7: All local iterations are completed.

Figure 3 illustrates how the *extrinsic* LLR values are passed among the decoders for the 2-dimensional SPC-PC local decoder.

C. Output Extrinsic Message

After a fixed number of *local* turbo iterations ($\tau = \tau_{max}$) have been carried out, the overall *extrinsic* LLR value for a particular bit indexed by (a, b) is obtained by summing the corresponding *extrinsic* LLR values in all dimensions, i.e.,

$$\mathbb{E}_{a,b} = \sum_{c=1}^{D} \mathbb{E}_{a,b,c} \quad (11)$$

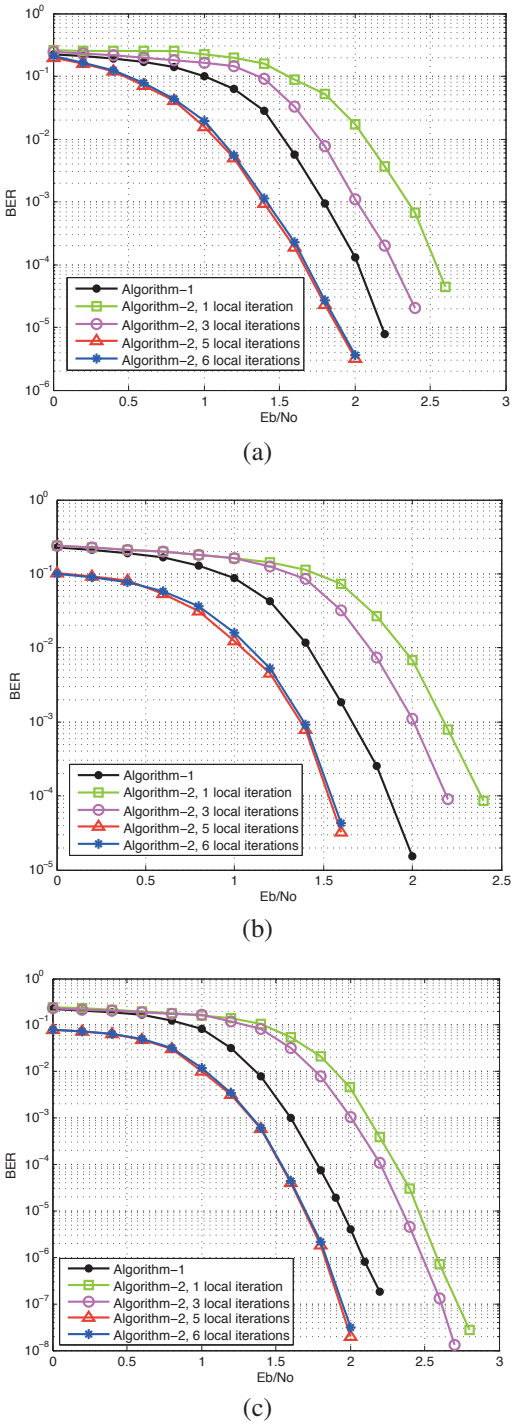
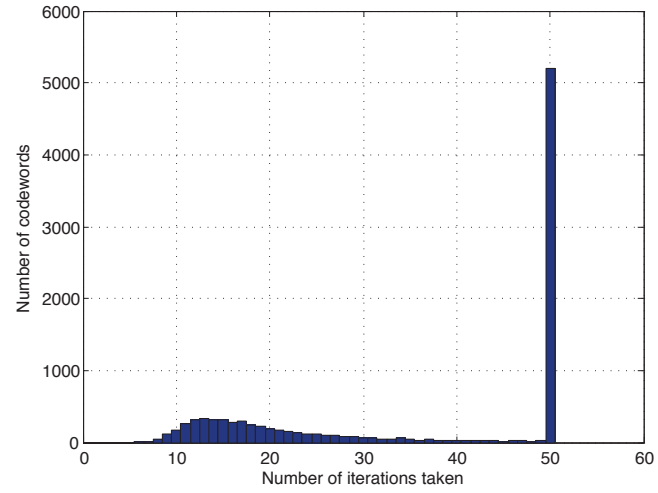


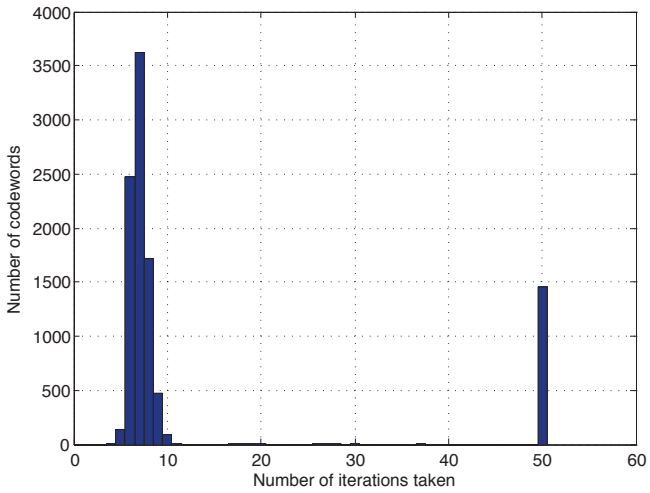
Fig. 4: Bit error rate of DGLDPC-1 when the maximum number of *global* iterations equals (a) 10; (b) 20; and (c) 50.

where $a = 1, 2, ..., (n_{spc})^{D-1}$ and $b = 1, 2, ..., n_{spc}$. Finally, these overall *extrinsic* LLR values are passed to the connected SVNs.

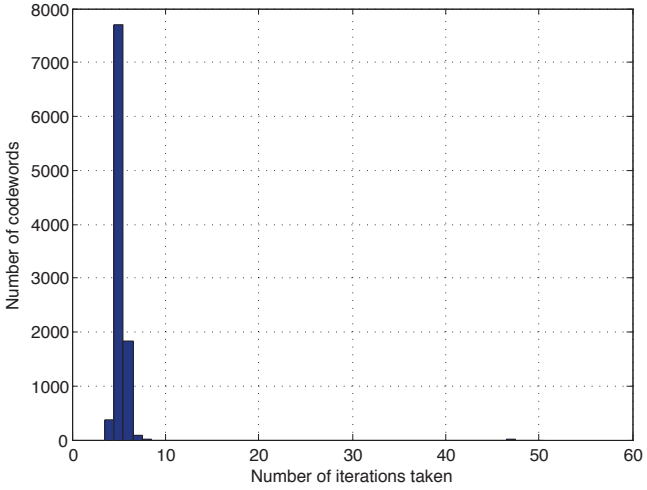
The decoder in each SVN, based on all incoming messages, evaluates the *extrinsic* LLR values. Afterward, the messages are returned to the connected SCNs and one *global* iteration



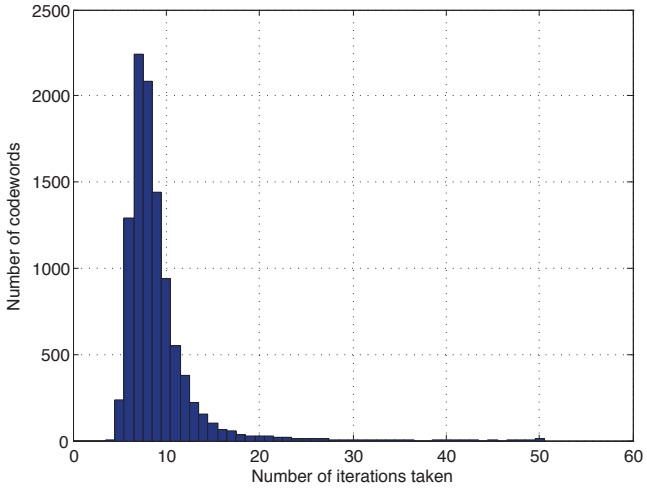
(a) Convergence rate of DGLDPC-1 with 1 *local* turbo iteration.



(b) Convergence rate of DGLDPC-1 with 3 *local* turbo iterations.



(c) Convergence rate of DGLDPC-1 with 5 *local* turbo iterations.



(d) Convergence rate of DGLDPC-1 with the algorithm in [17].

Fig. 5: Convergence rate of DGLDPC-1 with different decoding algorithms. (a)–(c) Algorithm-2; (d) Algorithm-1. A maximum of 50 iterations is used for decoding each codeword.

is completed. When a sufficient number of *global* iterations are performed, the SVNs decode the codeword.

IV. Simulation Results

We use two different decoding algorithms in our simulations. Algorithm-1 denoting the algorithm from [17] is used as the baseline for performance comparison and that Algorithm-2 is our proposed algorithm described in Section III. Note that Algorithm-1 is based on the MAP algorithm and no iteration is needed at the SCN decoder.

A. Error Performance

A DGLDPC code denoted as DGLDPC-1 is constructed with the following parameters.

- $(4, 3)$ SPC codes as constituent codes in all SVNs
- $(4, 3)^2$ SPC-PCs as constituent codes in all SCNs
- Adjacency matrix of size $M_{a,1} \times N_{a,1} = 250 \times 1000$
- Code rate 0.417
- Code length 3000

We assume a binary input-additive white Gaussian noise (BI-AWGN) channel. Figure 4 depicts the bit-error-rate (BER) performance when DGLDPC-1 is decoded with a maximum of 10, 20 and 50 *global* iterations, respectively. Both Algorithm-1 and Algorithm-2 have been used at the SCNs. For Algorithm-2, the number of *local* iterations used is set to 1, 3, 5 and 6. From the curves, it can be observed that for a fixed number of *local* iterations, the error performance of the DGLDPC-1 code improves with the maximum number of *global* iterations. When the maximum number of *global* iterations is fixed, we can see that the error performance improves when the number of *local* iterations performed by Algorithm-2 at the SCNs increases from 1 to 3, and then to 5. However, if the number of *local* iterations is increased to 6, the performance does not improved anymore. In fact, we find that using iterations more than 6 will further degrade the error performance. We also observe that Algorithm-2 using 5 *local* iterations outperforms Algorithm-1 [17]. For example, when a maximum of 50 *global* iterations are used, Algorithm-2 outperforms Algorithm-1 by

TABLE I: Simulation statistics of the DGLDPC-1 code.

Codes	No. of converged codewords	Total time consumed	Total no. of *global* iters.	Avg. time taken per *global* iter.	Avg. no. of global iters. per codeword
DGLDPC-1, 1 *local* turbo iters.	4803	1477 seconds	355410	0.0042 seconds	35.54
DGLDPC-1, 3 *local* turbo iters.	8540	1957 seconds	133369	0.0147 seconds	13.33
DGLDPC-1, 5 *local* turbo iters.	10000	2222 seconds	51706	0.0430 seconds	5.17
DGLDPC-1, algorithm in [17]	9993	20138 seconds	89054	0.2261 seconds	8.99

about 0.3 dB at a BER of 2×10^{-7}. Besides, we do not observe any error floor at a BER of 2×10^{-8}.

B. Convergence Rate

We send 10000 DGLDPC-1 codewords, and then decode them by Algorithm-1 (Algorithm in [17]) and Algorithm-2 at $E_b/N_0 = 1.6$ dB. We also set the maximum number of *global* iterations to 50. The number of *global* iterations taken to decode each codeword is recorded and the statistical data are plotted in Figure 5. From the figure, we can easily find that Algorithm-2 using 1 or 3 *local* iterations does not perform as well as Algorithm-1 because many of the codewords cannot be decoded even after 50 *global* iterations have been performed. On the other hand, Algorithm-2 using 5 *local* iterations outperforms Algorithm-1 because all the codewords can be decoded with a very small number of *global* iterations. Such observations are consistent with those made in the previous section. In Table I, we list the simulation times. We can see that Algorithm-2 using 5 *local* iterations requires about $1/9$ of the simulation time compared with Algorithm-1.

V. Conclusion

In this paper, we have proposed a class of DGLDPC codes based on SPC-PCs as component codes at SCNs and SPCs at SVNs. We have also proposed a decoding algorithm (Algorithm-2) that uses an iterative decoder at the SCNs. Simulation results have shown that the proposed DGLDPC code possesses excellent error performance. Moreover, Algorithm-2 with an appropriate number of *local* iterations can outperform the decoding algorithm in [17] in terms of both speed and error performance.

Acknowledgements

The work described in this paper was partially supported by a grant from the RGC of the Hong Kong SAR, China (Project No. PolyU 5190/11E).

References

[1] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[2] G. Liva and W. E. Ryan, "Short low-error-floor Tanner codes with Hamming nodes," *in Proc. IEEE Milcom*, pp. 208–213, Oct. 2005.

[3] J. Boutros, O. Pothier, and G. Zemor, "Generalized low density (Tanner) codes," in *Proc. IEEE. Int. Conf. Commun.*, vol. 1, Jun. 1999, pp. 441–445.

[4] M. Lentmaier and K. Zigangirov, "On generalized low-density parity-check codes based on Hamming component codes," *IEEE Commun. Letters*, vol. 3, no. 8, pp. 248–250, Aug. 1999.

[5] N. Miladinovic and M. Fossorier, "Generalized LDPC codes with Reed-Solomon and BCH codes as component codes for binary channels," in *Proc. IEEE GLOBECOM*, vol. 3, Nov. 2005, p. 6.

[6] I. Djordjevic, L. Xu, T. Wang, and M. Cvijetic, "GLDPC codes with Reed-Muller component codes suitable for optical communications," *IEEE Commun. Letters*, vol. 12, no. 9, pp. 684–686, Sep. 2008.

[7] A. S. Shadi, G. Liva, and W. E. Ryan, "Low-floor Tanner codes via Hamming-node or RSCC-Node doping," in *Proc. the 16th international conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, 2006, pp. 245–254.

[8] J. Chen and R. M. Tanner, "A hybrid coding scheme for the Gilbert-Elliott channel," *IEEE Trans. Commun.*, vol. 54, pp. 1787–1796, 2006.

[9] P. Elias, "Error free coding," *IRE Trans. Inform. Theory*, vol. 4, pp. 29–37, Sept. 1954.

[10] D. Rankin, T. Gulliver, and D. Taylor, "Asymptotic performance of single parity check product codes," *IEEE Trans. on Inform. Theory*, vol. 49, no. 9, pp. 2230–2235, Sep. 2003.

[11] J. Lodge, P. Hoeher, and J. Hagenauer, "Separable MAP 'filters' for the decoding of product and concatenated codes," in *Proc. IEEE Int. Conf. Commun.*, 1993, pp. 1740–1745.

[12] D. Rankin and T. Gulliver, "Single parity check product codes," *IEEE Trans. on Commun.*, vol. 49, no. 8, pp. 1354–1362, Aug. 2001.

[13] A. Barg and G. Zemor, "Distance properties of expander codes," *IEEE Trans. Inform. Theory*, vol. 52, no. 1, pp. 78–90, Jan. 2006.

[14] B. Sklar, *Digital Communications: Fundamentals and Applications*. second edition, Prentice Hall, 2004.

[15] A. Neubaue, J. Freudenberger, and V. Kuhn, *Coding Theory: Algorithms, Architectures and Application*. John Wiley and Sons, 2007.

[16] F. L. Y. Min and C. Tse., "Generalized LDPC code with single-parity-check product constraints at super check nodes," in *7th International Symposium on Turbo Codes and Iterative Information*, Aug. 2012, pp. 165–169.

[17] Y. Wang and M. Fossorier, "Doubly generalized LDPC codes over the AWGN channel," *IEEE Trans. Commun.*, vol. 57, no. 5, pp. 1312–1319, May 2009.