Encoding of Spatially Coupled LDGM Codes for Lossy Source Compression

Ahmad Golmohammadi, *Student Member, IEEE,* David G. M. Mitchell, *Senior Member, IEEE,* Jörg Kliewer, *Senior Member, IEEE*, and Daniel J. Costello, Jr., *Life Fellow, IEEE*

***Abstract*—It has been shown that a class of spatially coupled low-density generator-matrix (SC-LDGM) code ensembles displays distortion saturation for the lossy binary symmetric source coding problem with the belief propagation guided decimation (BPGD) algorithm, *i.e.*, the BPGD distortion approaches the optimal expected distortion of the underlying ensemble asymptotically in code length. Here, we investigate the distortion performance of a practical class of protograph-based SC-LDGM code ensembles and demonstrate distortion saturation numerically. Moreover, taking advantage of the convolutional structure of the SC-LDGM codes, we propose an efficient windowed encoding (WE) algorithm with two decimation techniques for lowering the WE complexity that maintain distortion performance close to the rate-distortion bound.**

***Index Terms*—Lossy source coding, spatial coupling, LDGM codes, belief propagation guided decimation, rate distortion bound.**

I. Introduction

Compared to lossless compression, where the source must be reconstructed identically from its compressed version, the *lossy* source compression problem requires the reconstructed data to be correct only up to some specified distortion measure. Applying linear codes for lossy source compression of discrete sources is a classical idea [1], and various bounds on distortion performance have been proposed for different types of linear codes, *e.g.*, trellis codes [2], [3], turbo-codes [4], LDPC codes [5], and polar codes [6], [7], [8]. Excellent reconstruction performance can be obtained with such approaches. For example, polar codes were shown to have optimal lossy source coding performance for the binary symmetric source [8], and it was shown that it is possible to approach the binary *rate-distortion* (RD) limit using LDPC codes if the node degrees of the ensemble grow logarithmically with the block length [5]. Moreover, compound LDPC and *low-density generator matrix* (LDGM) codes have been shown to achieve the RD limit with bounded node degrees [9].

Low-density generator matrix block codes (LDGM-BCs), the duals of *LDPC block codes* (LDPC-BCs), were shown to achieve the RD limit under optimal encoding as the average node degrees increase [10], and lower bounds on the distortion have been derived both for random ensembles [11] and for specific code constructions [12]. Like LDPC-BCs, LDGM-BCs can be defined on a sparse graph and are amenable to low complexity message passing algorithms. Unfortunately, this approach typically fails because the lossy source coding problem has multiple optimal (or near-optimal) solutions and one cannot find the relevant fixed point without reducing the solution space. Heuristic decimation algorithms based on *belief propagation* (BP) [10], [13], [14], [15] and survey propagation [10], [16], [17], [18] have been proposed for lossy data compression for both linear and non-linear codes and have been successfully applied to k-satisfiability (k-SAT) constraint problems [13], [19], [20]. Also, good LDGM-BCs have been designed for the binary erasure source [21], [22], [23]; however, to the best of our knowledge, there have been no constructive LDGM-BC designs proposed with performance guarantees for general binary discrete sources. Indeed, the best known code designs are heuristic in nature and typically use dual codes to LDPC-BCs that were optimized for the binary symmetric channel (see, *e.g.*, [10], [24], [25]).

Spatially coupled LDGM (SC-LDGM) codes can be obtained by coupling together (connecting) a series of L LDGM-BC graphs to make a larger connected graph with a convolutional-like structure. In [26], [27], a *belief propagation guided decimation* (BPGD) algorithm was applied for lossy source compression of the binary symmetric source to a class of SC-LDGM codes with regular check node degrees and both Poisson distributed [26] and regular [27] variable node degrees. It was demonstrated there that the SC-LDGM code ensembles achieve *distortion saturation*, in the sense that the distortion of the SC-LDGM code ensemble approaches the optimal distortion for the underlying LDGM-BC ensemble as the coupling length and code length tend to infinity, which, in turn, approaches the RD limit as the node degrees increase. As a result, SC-LDGM codes have great promise for the lossy source coding problem, but the large code lengths required for distortion saturation with the standard BPGD algorithm make them unattractive in practice.

Motivated by the theoretical results of [26], [27], this paper investigates several fundamental practical issues related to SC-LDGM codes. In particular, the main contributions are as follows:

1) We present a practically interesting (J, K)-regular SC-LDGM code construction based on *protographs*, which are amenable to efficient implementation (see, *e.g.*,

This material is based upon work supported by the National Science Foundation under Grant Nos. CCSS-1710920, CCF-1440001, and ECCS-1711056. This paper was presented in part at the 2016 IEEE International Symposium on Information Theory and in part at the 2016 IEEE International Symposium on Information Theory and Its Applications.

A. Golmohammadi and D. G. M. Mitchell are with the Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM 88003, USA (e-mail: {golmoham, dgmm}@nmsu.edu).

J. Kliewer is with the Dept. of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA (e-mail: jkliewer@njit.edu).

D. J. Costello, Jr. is with the Dept. of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA (e-mail: costello.2@nd.edu).

[28]). We demonstrate distortion saturation numerically for these protograph-based code ensembles and show that this effect is the result of a "wave-like" encoding behavior. Note that, even for the well studied SC-LDPC channel coding problem, it has not been proven that protograph-based SC-LDPC codes achieve threshold saturation and channel capacity. Thus demonstrating that practically interesting protograph-based SC-LDGM codes empirically achieve distortion saturation is a valuable result;

2) To reduce the latency and thus combat the need for very long code lengths, as is desirable in practice, we propose a novel windowed encoding (WE) scheme and demonstrate distortion performance close to the RD limit with moderate encoding latency for the binary symmetric source. To the best of our knowledge, regular SC-LDGM codes with the proposed WE scheme are the first regular LDGM constructions that perform close to the RD limit with low complexity encoding and moderate latency;
3) We propose a combined soft and hard decimation algorithm that allows the wave-like encoding to progress through the chain and reduces the average distortion compared to soft only decimation algorithms. Moreover, we present two further decimation approaches for WE that can significantly lower computational complexity (measured as the number of node updates). The first decimates a fixed number of nodes per iteration and the second decimates all nodes with sufficiently large confidence at each iteration. For both approaches, we study the trade-offs in terms of performance vs. complexity and show that the complexity can be significantly decreased with little loss in performance compared to decimating a single node per iteration;
4) We discuss the latency, complexity, and performance trade-offs of WE and compare the results with an optimized irregular LDGM-BC. Moreover, we demonstrate that a min-sum formulation can be used during the BP updates that further reduces the complexity at the cost of a slight loss in performance;
5) Finally, we demonstrate robust performance over a variety of compression rates for different (J, K)-regular SC-LDGM codes. Since there are many parameters involved in the code construction and encoding algorithms, one of our goals is to explain their effect on performance with respect to the distortion/latency/complexity trade-offs and to give a system designer guidance in choosing reasonable parameter sets for implementation.

The reminder of the paper is organized as follows. Section II contains necessary background material on lossy source coding and introduces protograph-based SC-LDGM codes. In Section III, the proposed source compression algorithm for SC-LDGM codes is presented along with experimental results showing distortion saturation and a discussion of the wave-like encoding behavior. In Section IV, we present the windowed encoding algorithms, including a study of the distortion/latency/complexity trade-offs. We conclude the paper in Section V.

II. SC-LDGM Code Ensembles for lossy compression

A. Lossy Source Compression

In this paper, we consider compressing the *symmetric Bernoulli source* [29], where the source sequence $\mathbf{s} = (s_1, s_2, ..., s_n) \in F_2^n$ consists of independent and identically distributed (i.i.d.) random variables with $\mathbb{P}(s_i = 1) = 1/2$, $i = 1, 2, \ldots, n$. We wish to represent a given source sequence by some codeword $\mathbf{z} \in F_2^m$ from a given code C containing $2^m = 2^{nR}$ codewords, where $m \ll n$ and $R = \frac{m}{n}$ is the *compression rate*. The codeword $\mathbf{z}$ is used to reconstruct the source sequence as $\hat{\mathbf{s}}$, where the mapping $\mathbf{z} \to \hat{\mathbf{s}}(\mathbf{z})$ depends on the code C. The quality of reconstruction is measured by a *distortion metric* $d : \mathbf{s} \times \hat{\mathbf{s}} \to \mathbf{R}^+$ and the resulting source encoding problem is to find the codeword with minimal distortion, *i.e.*, $\hat{\mathbf{z}} = \arg\min_{\mathbf{z}} d(\mathbf{s}, \hat{\mathbf{s}})$. For a symmetric Bernoulli source, the typical measure of distortion is the *Hamming metric* $d_H(\mathbf{s}, \hat{\mathbf{s}}) = \frac{1}{n} \sum_{i=1}^{n} |s_i - \hat{s}_i|$, and thus the average quality of the reconstruction is measured as $\bar{D} = \mathbb{E}_{\mathbf{s}}[d_H(\mathbf{s}, \hat{\mathbf{s}})]$. For any encoding scheme, the average distortion is bounded below by *Shannon's RD limit* D_{Sh}, defined implicitly by the rate-distortion function $h(D_{Sh}) = 1 - R, D_{Sh} \in [0, 0.5]$, where $h(\cdot)$ is the binary entropy function [30]. The goal is to find an encoding scheme that achieves the rate-distortion bound for a given R.

B. Protograph-based SC-LDGM Codes

An LDGM-BC ensemble can be represented by means of a *protograph* [31], a small bipartite graph that connects a set of n source (information) and m code (compressed) nodes to a set of n generator nodes by a set of edges. It can be represented by a generator or *base* biadjacency matrix $\mathbf{B}$, where $B_{x,y}$ is taken to be the number of edges connecting generator node g_y to code node z_x. The generator matrix $\mathbf{G}$ of a protograph-based LDGM-BC can then be created by expanding $\mathbf{B}$ using a *lifting factor* M, where each non-zero entry in $\mathbf{B}$ is replaced by a sum of $B_{x,y}$ non-overlapping permutation matrices of size $M \times M$ and each zero entry is replaced by the $M \times M$ all-zero matrix. Fig. 1 depicts the protograph representation of a $(3, 6)$-regular LDGM-BC ensemble with the all-ones base matrix $\mathbf{B}$ of size 3×6. The white and black circles represent source (information) and code (compressed) symbols, respectively. After compression, the reconstructed source symbols $\hat{s}_i$ are obtained by a modulo 2 summation at the generator nodes.

Protograph-based SC-LDGM codes are constructed by *coupling* together a series of L disjoint, or uncoupled, LDGM-BC graphs into a single coupled chain. Starting from a $b_c \times b_r$ block base matrix $\mathbf{B}$, an "edge-spreading" construction [32]

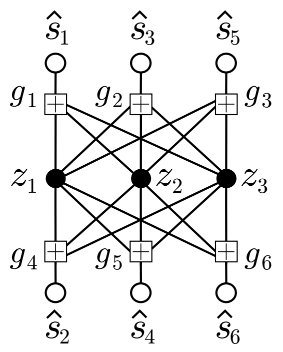


Fig. 1. Protograph representing a (3, 6)-regular LDGM-BC ensemble.

can be used to form the base matrix of an SC-LDGM code ensemble with *coupling length* L as

$$\mathbf{B}_{[0,L-1]} = \begin{bmatrix} \mathbf{B}_0 & & & \\ \mathbf{B}_1 & \mathbf{B}_0 & & \\ \vdots & \mathbf{B}_1 & \ddots & \\ \mathbf{B}_w & \vdots & \ddots & \mathbf{B}_0 \\ & \mathbf{B}_w & & \mathbf{B}_1 \\ & & \ddots & \vdots \\ & & & \mathbf{B}_w \end{bmatrix}_{(L+w)b_c \times L b_r}, \quad (1)$$

where $\mathbf{B}_0 + \mathbf{B}_1 + \cdots + \mathbf{B}_w = \mathbf{B}$, w denotes the *coupling width*, and the $b_c \times b_r$ matrices $\mathbf{B}_i$ are referred to as *component base matrices*, $i = 0, 1, \ldots, w$. An ensemble of SC-LDGM codes can then be formed from $\mathbf{B}_{[0,L-1]}$ using the protograph construction method described above. The *design compression rate* of the ensemble of SC-LDPC codes is

$$R_L = \left(1 + \frac{w}{L}\right) \frac{b_c}{b_r}, \quad (2)$$

where we note that R_L is monotonically decreasing and approaches b_c/b_r as $L \to \infty$.

For convenience we focus most of our attention on $(J, 2J)$-regular SC-LDGM code ensembles, where $\mathbf{B} = [J, J]$ and the component base matrices are chosen as $\mathbf{B}_0 = \mathbf{B}_1 = \cdots = \mathbf{B}_w = [1, 1]$ for $w = J - 1$, although some results for general (J, K)-regular ensembles are given in Section IV-B, and an extension to irregular ensembles is straightforward. In order to improve distortion performance for BP algorithms, as in [26] we found it necessary to reduce the generator node degrees at the start of the chain (this will be explained in Section III). This was achieved for the $(J, 2J)$-regular ensembles by deleting the first $J - 2$ rows of (1).[1] The resulting ensembles are denoted SC$(J, 2J)$ and have modified design compression rate

$$R_L = \left(1 + \frac{w - J + 2}{L}\right) \frac{b_c}{b_r} \xrightarrow[L \to \infty]{} \frac{b_c}{b_r}. \quad (3)$$

[1]Note that removing code nodes by deleting rows of the protograph actually makes the encoding problem harder, since the compression rate decreases. However, as we show in Section III-C5, the reduced generator node degrees and associated improved convergence of the algorithm at the start of the chain more than compensates for the added difficulty of encoding.

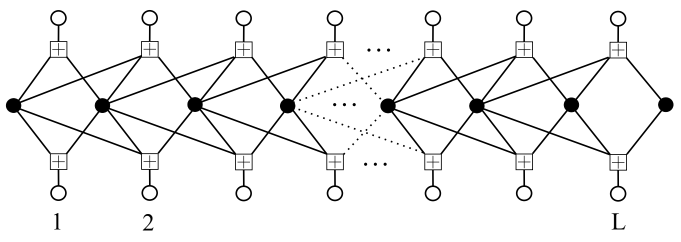


Fig. 2. Protograph of the (3, 6)-regular SC-LDGM ensemble SC(3, 6).

Fig. 2 depicts the protograph representation of the SC(3, 6) SC-LDGM code ensemble with coupling length L and coupling width $w = J - 1 = 2$.

III. Source Reconstruction Using SC-LDGM Codes

In this section, we first review some relevant message passing decimation algorithms in Section III-A. We then introduce a new "soft-hard" decimation algorithm in Section III-B and investigate its encoding dynamics and performance in Section III-C.

A. Message Passing Decimation Algorithms

Although the sum product algorithm (SPA) can achieve excellent performance for channel coding problems, directly applying it to the lossy source coding problem does not give satisfactory results, since the approximate marginal calculated by the SPA does not provide reliable information for optimal encoding [10]. Modified BP algorithms, which include a *decimation* step, have been proposed and can be characterized as hard-decimation [10] or soft-decimation [25] algorithms. According to [10], hard-decimation involves two phases: 1) message passing, which approximates the marginal distributions, and 2) decimation, where some code nodes are assigned values and the graph is reduced by removing decimated code nodes. The procedure runs until the algorithm assigns a value to all code nodes. In soft-decimation [25], the decimation process is performed by modifying the BP equations at each iteration to assign appropriate *bias* values to the code nodes, whereby large bias values effectively decimate nodes but the graph is not reduced. The BP update equations for soft decimation at iteration t are given as follows [25]:

Code to generator node messages:

$$R_i^{(t+1)} = \sum_{a \in G(i)} R_{a \to i}^{(t)}, \quad R_{i \to a}^{(t+1)} = \sum_{b \in G(i) \backslash a} R_{b \to i}^{(t)}; \quad (4)$$

Generator to code node messages:

$$R_{a \to i}^{(t+1)} = \frac{1}{\mu} R_{i \to a}^{(t)} + 2(-1)^{s_a + |Z(a)|} \tanh^{-1} \left(\beta \prod_{j \in Z(a) \backslash i} B_{j \to a}^{(t)} \right); \quad (5)$$

Code node bias update:

$$B_i^{(t)} = \tanh\left(\frac{R_i^{(t)}}{2}\right), \quad B_{i \to a}^{(t)} = \tanh\left(\frac{R_{i \to a}^{(t)}}{2}\right), \quad (6)$$

where $R^{(t)}_{i\to a}$, $R^{(t)}_{a\to i}$, and $B^{(t)}_{i\to a}$ denote the message sent from code node i to generator node a, the message sent from generator node a to code node i, and the bias associated with $R^{(t)}_{i\to a}$ at iteration t, respectively, and β and μ are parameters. Then $R^{(t)}_i$ and $B^{(t)}_i$ denote the likelihood ratio of code node i and the bias associated with $R^{(t)}_i$, repectively. Also, for an LDGM graph Γ with code nodes Z and generator nodes G, $Z(a)$ denotes the set of all code node indices connected to generator node a, $|Z(a)|$ is its cardinality, and $G(i)$ denotes the set of all generator node indices connected to code node i, $\forall i \in Z$ and $\forall a \in G$. Without the term $\frac{1}{\mu}R^{(t)}_{i\to a}$ in (5), equations (4)-(6) are equivalent to BP with no decimation. Including the first term in (5) incorporates soft-decimation via the "*soft-indicator*" *function* [25] $I_{soft}(B^{(t)}_{i\to a}) = \frac{2}{\mu}\tanh^{-1}(B^{(t)}_{i\to a}) = \frac{1}{\mu}R^{(t)}_{i\to a}$, which approximates the "*hard-indicator*" *function*

$$I_{hard}\left(B^{(t)}_{i\to a}\right) = \begin{cases} -\infty & B^{(t)}_{i\to a} = -1 \\ 0 & -1 < B^{(t)}_{i\to a} < 1, \\ +\infty & B^{(t)}_{i\to a} = 1 \end{cases} \tag{7}$$

where μ controls the "softness" of the approximation. For further details, see [25]. Since β and μ are free parameters, they can be optimized independently; however, here we choose them as $\beta = (1-\xi)/(1+\xi)$ and $\mu = 1/\xi$ for simplicity, where ξ is called the *tuning parameter*. This choice simplified the optimization and was shown to yield good numerical results in [25].

B. *Soft-Hard Decimation*

The BP algorithm used in this paper is a mixture of both hard and soft decimation. The algorithm uses the soft decimation equations given above, but after each iteration it searches for a code node with maximum bias value. This code node is then decimated (permanently fixed) and the current graph reduced following the hard decimation rule. This modification is applied to force decimation to occur in designated areas of the code graph in order to take advantage of the convolutional structure of SC-LDGM codes. The procedure is described in detail as Algorithm A, where t indicates the iteration number, $\Gamma^{(t)}$ is the LDGM code graph at iteration t, and z_i represents the binary value assigned to code node i. The initial code to generator node messages, $R^{(0)}_{i\to a}$, are set to ± 0.1 with $\mathbb{P}(R^{(0)}_{i\to a} = 0.1) = 0.5$, and reset to 0 at iteration 1.

Algorithm A: Soft-Hard Decimation

```
1) At iteration $t = 0$, initialize graph instance $\Gamma^{(t=0)}$
2) Update equations (4), (5), and (6)
3) Find the maximum bias $B^{(t)} = \max_i \{|B^{(t)}_i| \mid i$ not fixed$\}$
4) If $B^{(t)} > 0$ then
         $z_i \leftarrow '1'$
   else
         $z_i \leftarrow '0'$
5) Decimate graph as
     a) $\forall a \in G(i)$, $s_a \leftarrow s_a \oplus z_i$ (update source symbols)
     b) reduce the graph as $\Gamma^{(t+1)} = \Gamma^{(t)} \backslash \{i\}$ (remove code
        node $i$ and all its edges)
6) If there exist any unassigned code symbols $z_i$
         go to 2)
   else
         exit algorithm and return code symbols
```

C. *Analysis and Dynamics of Encoding SC-LDGM Codes*

In this subsection, the results of various experiments of a C++-based implementation of Algorithm A are reported for different SC-LDGM code parameters. Codes were obtained by randomly lifting the protograph and all results were obtained by averaging over 1000 trials.

1) Effect of the tuning parameter ξ: Fig. 3 shows the average distortion versus ξ for SC(3,6), SC(4,8), and SC(5,10) codes with code length $n = 102400$, coupling length $L = 100$, and compression rate $R_L = 0.5050$. By running the proposed algorithm for SC(3,6), SC(4,8), and SC(5,10) codes, choosing the (experimentally determined to three decimal places) optimum ξ, denoted ξ^*, and averaging over 1000 trials,

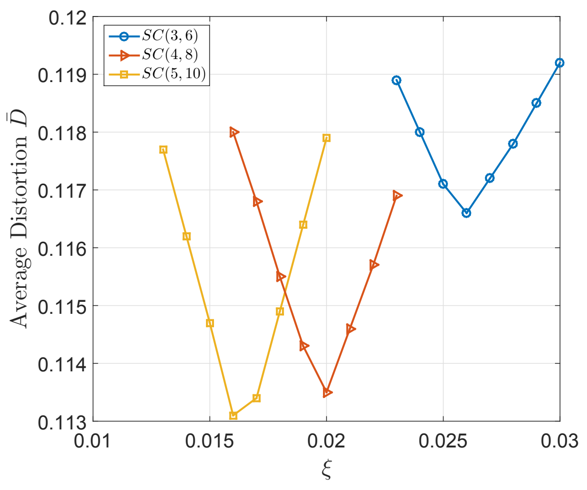


Fig. 3. The performance of Algorithm A as a function of ξ for SC(3,6), SC(4,8), and SC(5,10) code with length $n = 102400$, coupling length $L = 100$, and compression rate $R_L = 0.5050$.

average distortion values $\bar{D} = 0.116609$, $\bar{D} = 0.113463$, and $\bar{D} = 0.113137$ are obtained, respectively. Fig. 3 shows that the performance of the algorithm depends on the choice of ξ. We note that the optimum value of ξ for the SC(3,6), SC(4,8), and SC(5,10) codes is different and that by increasing the encoder node degrees from 3 to 5, the optimum value of ξ decreases. Moreover, Fig. 3 indicates that the sensitivity of the algorithm with respect to ξ, *i.e.*, the sharpness of the minima tends to increase with increasing density J.

2) Effect of increasing the coupling length L: We now consider SC-LDGM codes with fixed lifting factor $M = 512$ and L ranging from 4 to 100. Fig. 4 presents the *average distortion gap to the RD limit* $\bar{\delta} \triangleq \bar{D} - D_{Sh}$ obtained with respect to codeword length for SC(3,6), SC(4,8), and SC(5,10) codes. It can be seen from Fig. 4 that the average distortion gap decreases roughly exponentially with L and saturates to a value close to the RD limit; moreover, for fixed M, the gap decreases with increasing density J.[2]

[2]Note that the compression rate varies according to (2). Each point was compared to the RD limit for the given rate.

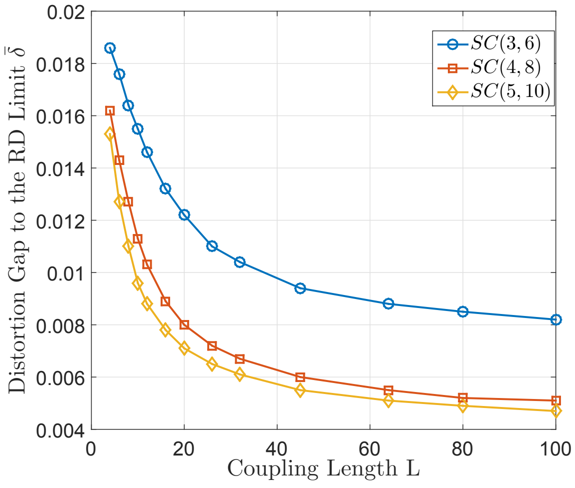


Fig. 4. Average distortion gap to the RD limit obtained for SC(3, 6), SC(4, 8), and SC(5, 10) codes with fixed $M = 512$ and increasing L.

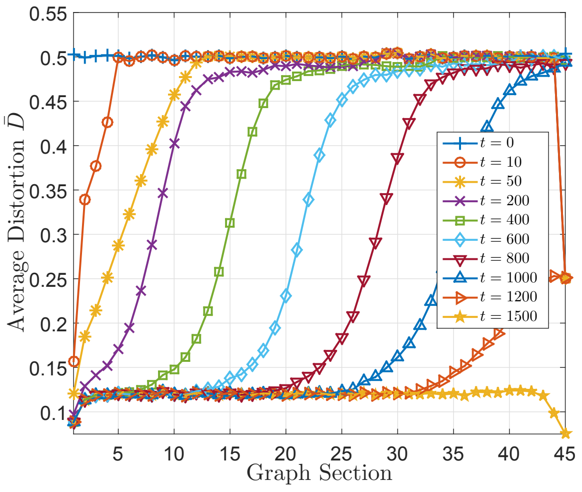


Fig. 5. Average distortion evolution of an SC(3, 6) code with $L = 45$ and $M = 512$ for various numbers of iterations of Algorithm A.

TABLE I
EFFECT OF THE LIFTING FACTOR M ON SC(4, 8) CODES IN TERMS OF AVERAGE DISTORTION $\bar{D}$ AND AVERAGE GAP TO THE RD LIMIT $\bar{\delta}$ WITH FIXED $L = 16$ AND COMPRESSION RATE $R_L = 0.5313$.

Lifting factor (M)	$\bar{D}$	$\bar{\delta}$
128	0.112675	0.0128
192	0.110917	0.0110
256	0.110074	0.0102
320	0.109571	0.0097
640	0.108645	0.0087
832	0.108478	0.0086
1024	0.108396	0.0085
1760	0.108225	0.0083
2048	0.108168	0.0082

3) Effect of increasing the lifting factor M: Table I shows the effect of increasing M. Here we consider SC(4, 8) codes with fixed $L = 16$, $R_L = 0.5313$, and RD limit $D_{Sh} = 0.09992$. We observe that, as expected, increasing M reduces the average distortion and we obtain saturating values that approach the RD limit. Since SC-LDGM codes experience saturation in their distortion values both for increasing L and M, combining these results (by letting $M \to \infty$ and $L \to \infty$) implies that SC(J, K) codes numerically approach the optimal distortion values of the underlying (J, K)-regular LDGM-BCs. For example, for a relatively large SC(4, 8) code with $L = 100$, $M = 512$, and $R_L = 0.5050$, we obtained an average distortion of $\bar{D} = 0.113463$, where the optimal average distortion of a (4, 8)-regular LDGM-BC of $R_L = 0.5$ is $\bar{D}_{opt} = 0.1111$ [27].

4) Wave-like encoding: Fig. 5 shows the average distortion evolution over each section of the graph of an SC(3, 6) code with $L = 45$, $M = 512$, and a total of 23552 code nodes for various numbers of iterations t of Algorithm A. Note that the behavior is similar to the wave-like decoding of SC-LDPC codes for channel coding [32], where the code nodes at the left end of the graph generate large bias values (reliable information, thereby starting the decimation process), which then propagate through the graph from left to right with increasing iterations. Initially, the average distortion is around 0.5, but after a few iterations the degree two generator nodes (at the left side of the graph) facilitate convergence of the attached code nodes, thereby generating large bias values and starting an "encoding wave" (see also the following subsection). Note that if we decimate a single code node per iteration, the algorithm requires 23552 iterations to terminate, but Fig. 5 shows that after decimating only 1500 code nodes, the bias values have already saturated and the algorithm can be stopped.

The wave-like encoding phenomenon and propagation of reliable bias values from left to right motivates the use of a windowed encoding scheme, where we modify the decimation progress during the encoding wave, thereby reducing both computational complexity and encoding latency. For example, the complexity can be reduced by decimating multiple code nodes per iteration. This could be done in a fixed way, *i.e.*, by decimating x nodes per iteration, or by using threshold values, *i.e.*, by decimating nodes only when a sufficiently large bias value is obtained. Both approaches are discussed in Section IV.

5) Effect of the generator node degrees at the ends of the code graph: As mentioned in Section II, we choose to modify the SC-LDGM protograph in (1) to reduce the generator node degrees at the left end of the graph to facilitate propagation of reliable information via wave-like encoding. In this section, we demonstrate how the choice of the generator node degrees affects the initialization and propagation of the encoding wave. As an example, we construct a (4, 8)-regular SC-LDGM base matrix in the form of (1) with $\mathbf{B}_0 = \mathbf{B}_1 = \mathbf{B}_2 = \mathbf{B}_3 = [1, 1]$ and consider deleting the first $j = 0, 1,$ and 2 rows. For $j = 0$, all generator nodes (columns of (1)) in the graph have degree 4, while the code nodes (rows of (1)) have degrees $2, 4, 6, 8, 8, \ldots$; for $j = 1$, the generator nodes in Section 1 (column 1 of (1)) have a reduced degree of 3 and the code node degrees are $4, 6, 8, 8, \ldots$; finally, for $j = 2$, the generator nodes in Sections 1 and 2 (columns 1 and 2 of (1)) have degrees 2 and 3, respectively, while the code nodes have degrees $6, 8, 8, \ldots$.

Fig. 6 shows snapshots of the encoding process with soft-hard decimation at $t = 500, 1500,$ and 3500 iterations (from

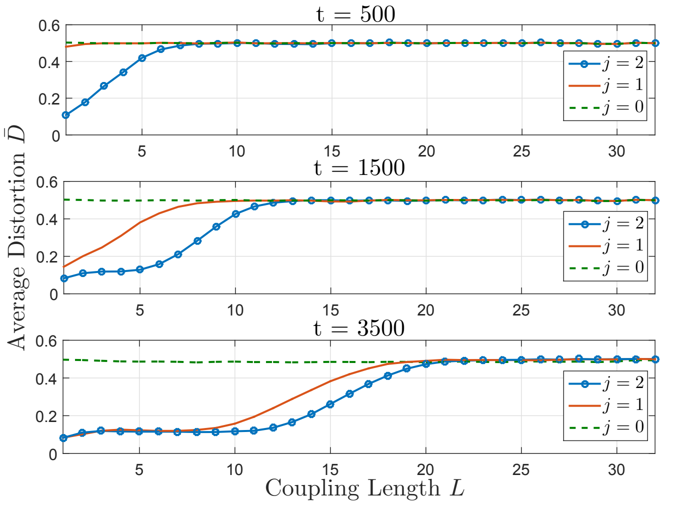


Fig. 6. Average distortion distribution among coupled sections of SC(4, 8) codes with $L = 32$, $M = 512$, and the first j rows deleted.

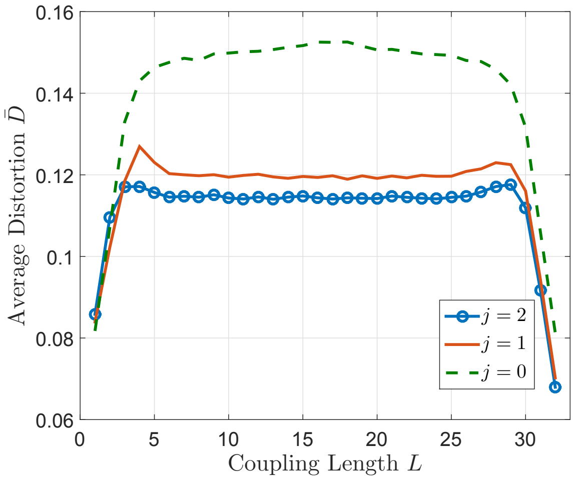
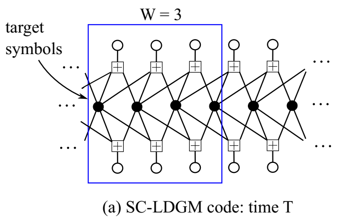


Fig. 7. Average distortion performance of SC(4, 8) codes with $L = 32$, $M = 512$, and the first j rows deleted.

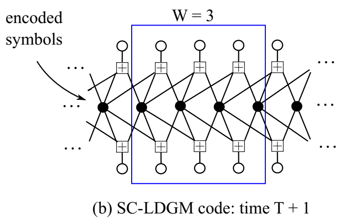
top to bottom) for the 3 constructions obtained with $j = 0, 1$, and 2 rows deleted and with code parameters $L = 32$ and $M = 512$. We observe that, by iteration $t = 500$, the average distortion of the leftmost sections of the $j = 2$ construction has decreased faster than the other constructions and the wave-like behavior has begun. On the other hand, the $j = 1$ construction shows only a slight improvement, while the $j = 0$ construction has not shown any improvement and has average distortion $\bar{D} = 0.5$ over all sections. By iteration $t = 1500$, the encoding of Section 5 of the $j = 2$ construction is almost complete (the code nodes in this section have almost reached their final values) and the wave has reached Section 12; meanwhile, for $j = 1$, we observe the distortion has now improved at the leftmost end and the wave-like behavior is visible; however, we still see no improvement for $j = 0$. Finally, by iteration $t = 3500$, we observe that the encoding wave for $j = 1$ and $j = 2$ are now steadily moving through the chain, with the $j = 2$ construction reaching further due to its earlier advantage; however, there is still no significant distortion improvement for $j = 0$.

The reduced generator nodes not only increase convergence/wave propagation speed, they also have an impact on the resulting distortion. Fig. 7 shows the average distortion results per graph section obtained for the $j = 0, 1$, and 2 constructions after encoding is completed. The poor performance of the $j = 0$ construction with soft-hard decimation can be explained by investigating the encoding progress across the graph sections at different iterations t. We saw in Fig. 6 that the distortion decreases slowly at the right end of the graph; however, algorithm A still enforces decimation at each iteration. We observed experimentally that roughly the first one-sixth of decimated code nodes occurred randomly throughout the graph with low reliability (bias values near 0). Eventually, though, the distortion decreases at the ends and the wave-like encoding begins; however, those initial decimated code nodes with low reliability result in a high overall distortion, as seen in Fig. 7. For $j = 1$, by around iteration $t = 1000$, the code nodes decimated by Algorithm A at the left end of the graph have reasonably good reliability (bias values about 0.7) and low distortion values; after that, the encoding wave proceeds, decimating code nodes with high reliability (bias values about 0.9). Note that the initial improvement in convergence compared to the $j = 0$ case significantly lowers the overall distortion as the encoding wave travels through the graph. For $j = 2$, the algorithm quickly converges, with distortion decreasing rapidly over the leftmost sections of the graph as code nodes are decimated with high reliability (bias values about 0.9) by around iteration $t = 40$. This generates an encoding wave that propagates through the graph from left to right. Thus the $j = 2$ construction outperforms the others with Algorithm A as a result of better "seeding" of the wave, with more reliable information generated at the ends of the graph. Moreover, as the wave reaches the right end of the (non-symmetric) graph, all nodes are decimated with high reliability and the distortion is low.

Although our primary interest is to take advantage of the convolutional structure of the code and track the propagation of the encoding wave from left to right with a sliding window (see Section IV), where the right end of the graph is not involved until the end of the encoding process, it is also interesting to investigate the effect of modifying the right end of the chain when using Algorithm A. Suppose we remove the first j_1 rows and the last j_2 rows of (1). In the previous $j = 2$ case, now denoted $\mathbf{j} = (j_1 = 2, j_2 = 0)$, the degree 4 generator nodes at the right do not create a wave (as we saw in the $j = 0$ case above) and a single encoding wave propagates from left to right, but this is not the case if we also reduce the generator degrees at the right, *i.e.*, we choose $j_2 > 0$. For example, consider $\mathbf{j} = (2, 2), (2, 1)$, and $(1, 1)$, for the $(4, 8)$-regular construction. In the $\mathbf{j} = (2, 2)$ and $\mathbf{j} = (1, 1)$ cases, encoding waves propagate from both sides of the graph (much like they do in the $j = 2$ and $j = 1$ cases above), but, interestingly, when the waves meet in the center there is disagreement and the average distortion increases, since this disagreement ripples out from the center back to the ends via undecimated nodes. In general, we find that symmetric codes



(a) SC-LDGM code: time T



(b) SC-LDGM code: time T + 1

Fig. 8. The WE procedure operating on the protograph of an SC(3, 6) code.

result in higher distortion for the reasons discussed above. In the $\mathbf{j} = (2, 1)$ case, the encoding waves start independently from both ends of the graph, but the left wave moves faster, resulting in higher distortion at the right side of the code graph. As an interesting aside, we note that this behavior is not observed in the case of BP decoding of SC-LDPC codes, where there are not multiple optimal solutions and the decoding waves are typically in agreement.

IV. Windowed encoding

For the practical implementations of SC-LDGM codes with large coupling length L, it is essential to reduce the encoding latency. To this end, we propose a novel sliding window encoder, where a window of size W (containing W sections of the graph) slides over the graph from left to right. This is a similar concept to the sliding window decoder for channel coding with SC-LDPC codes [33]. At each window position, a modified BP algorithm (*e.g.*, Algorithm A) is applied to the code nodes in the window and all their neighboring generator nodes in order to encode one set of code symbols, called *target* symbols. After encoding the set of target symbols (*i.e.*, when they are all decimated), the window slides one section to the right and again executes the modified BP algorithm to encode the next set of target symbols, using both the nodes in the window and some previously encoded target symbols. Fig. 8 shows the WE procedure on the protograph of a SC(3, 6) code with window size $W = 3$ (covering 3 graph sections, or $3M$ code symbols). Here $2M$ source symbols enter the window at each window position and M code symbols leave (are encoded). In this paper, we refer to the *latency* of a WE scheme as the number of source symbols in the window, *i.e.*, how many source symbols we need to process before we can encode a set of target symbols. Full details of WE with two modifications for efficiency, termed Algorithms B and C, follow in the next two sections.

A. Multiple Node Decimations per Iteration (Algorithm B)

To reduce the number of required iterations, thereby reducing complexity, we consider modifying the algorithm to decimate the $N \geq 1$ code nodes with the highest bias values. In order to generate reliable messages and low distortion at the left end of the graph (initial nodes), (4), (5), and (6) are applied for a fixed number of iterations, I_f, to build reliable bias values before decimation begins. Details of this process are given in Algorithm B. In addition to the previous notation, $W^{(t)}$ denotes the window at iteration t, $W_Z^{(t)}$ denotes the set of code nodes inside the window, $W_G^{(t)}$ denotes the set of generator nodes inside the window, and $T(W^{(t)})$ denotes the set of target symbols inside the window. Finally, we use $W_{Z_p}^{(t)}$ and $W_{Z_f}^{(t)}$ to denote the set of past code nodes (to the left of the window) and future code nodes (to the right of the window), respectively, and $W_{G_p}^{(t)}$ and $W_{G_f}^{(t)}$ are defined similarly. The initial code to generator node messages, $R_{i\to a}^{(0)}$, are set to ± 0.1, with $\mathbb{P}(R_{i\to a}^{(0)} = 0.1) = 0.5$ and reset to 0 at iteration 1.

Algorithm B: Multiple Node Decimation

```
1) At iteration $t = 0$, initialize the graph to $\Gamma^{(t=0)}$
2) $\forall i \in W_Z^{(t)}, a \in W_G^{(t)}$ update (4), (5), and (6) for $I_f - 1$ iterations such that
    a) there is no incoming or outgoing message $\forall a \in W_{G_f}^{(t)}$
    b) there are incoming messages $\forall i \in W_{Z_p}^{(t)}$ and $\forall a \in W_{G_p}^{(t)}$, but no outgoing messages to those nodes
3) $\forall i \in W_Z^{(t)}, a \in W_G^{(t)}$ update (4), (5), and (6) according to 2a) and 2b)
4) Find the $N$ largest bias values among the remaining target symbols $B^{(t)} = \{B_i^{(t)} | \ i$ not fixed, $i \in T(W^{(t)})\}$
5) For each of the $N$ selected biases $i$, do:
    a) if $B_i^{(t)} > 0$ then
           $z_i \leftarrow '1'$
       else
           $z_i \leftarrow '0'$
       end if
    b) decimate graph as
        i) $\forall a \in G(i)$, $s_a \leftarrow s_a \oplus z_i$ (update source symbols)
        ii) reduce the graph as $\Gamma^{(t+1)} = \Gamma^{(t)} \backslash \{i\}$ (remove code node $i$ and all its edges)
6) if there exist any unassigned code symbols $i \in T(W^{(t)})$, go to 3)
   else if all source symbols are not encoded, shift window to the next position, go to 3)
   else exit algorithm
   end if
```

1) Windowed encoding results for $N = 1$: In this section, we present numerical results for WE of SC(4, 8) codes with $N = 1$ and $I_f = 0$ as an example. (Similar results were also obtained for other values of J and K.) Note that, by setting $N = 1$, Algorithm B can be thought of as the application of Algorithm A within a window. Also, like Algorithm A, the performance of the WE algorithm is sensitive to the choice of ξ, which was determined numerically for each code ensemble.

We considered an SC(4, 8) code with $L = 100$, $M = 512$, rate $R_L = 0.5050$, and RD limit $D_{Sh} = 0.1084$. Applying Algorithm B (windowed encoding with $N = 1$) with $W = 10$ and $\xi = 0.02$, the average distortion is $\bar{D} = 0.1172$, while applying Algorithm A (block encoding) gives $\bar{D} = 0.1135$, indicating there is only a slight loss in performance for a sufficiently large W. Fig. 9 shows the average distortion gap to the RD limit $\bar{\delta}$ (adjusted for the compression rate) for several SC(4, 8) LDGM code lengths L and window sizes $W = 4, 5, \ldots, 16$, with fixed $M = 512$, where the encoding

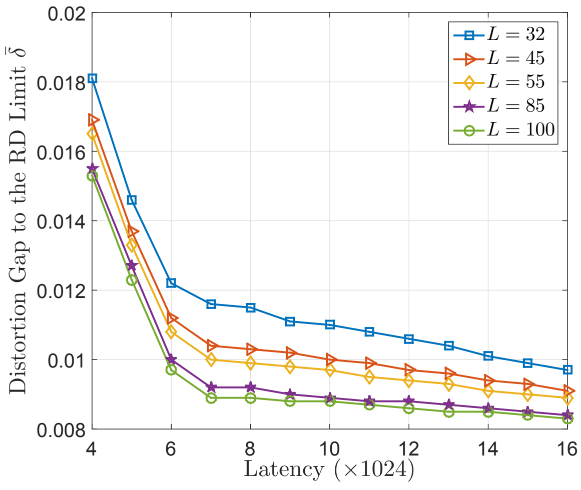


Fig. 9. Average distortion gap to the RD limit for various SC(4, 8) codes with increasing latency (window size W).

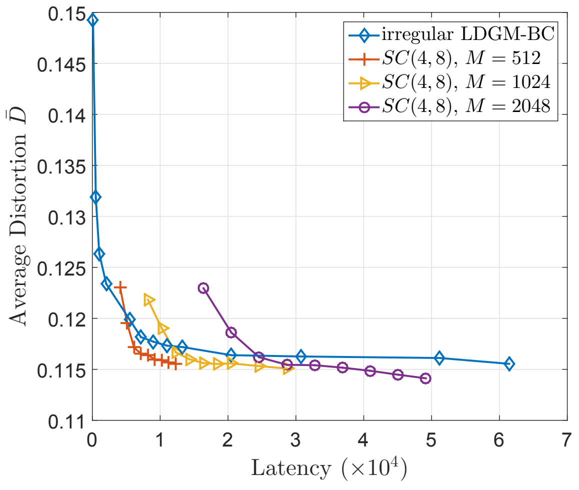


Fig. 10. Average distortion versus latency of (4, 8)-regular SC-LDGM codes with different lifting factors and $L = 32$ compared to optimized, irregular LDGM-BCs. Latency for the SC-LDGM code corresponds to the window size, whereas the latencies for the LDGM-BCs correspond to their block lengths.

latency is equal to $2MW$. We observe that by increasing the code length and latency, the average distortion saturates to a value close to the RD limit for the given rate. We also see that increasing W (latency) beyond a certain value improves the distortion only slightly. For example, we note that only moderate improvements in distortion are obtained by increasing W beyond 8.

Fig. 10 plots the average distortion as a function of latency obtained by applying Algorithm B (with $N = 1$) to SC(4, 8) codes with $L = 32$, compression rate $R = 0.5156$, and $W = 4, 5, \ldots, 12$ for $M = 512$, 1024, and 2048, respectively. Also plotted for comparison is the average distortion of an irregular LDGM-BC (optimized for the soft decimation algorithm) with compression rate $R = 0.5$ and degree distribution [25]

$$\lambda(x) = x^6, \tag{8}$$

$$\rho(x) = 0.275698x + 0.25537x^2 + 0.076598x^3 + 0.39233x^8. \tag{9}$$

Note that the latency of the LDGM-BC is its total block length, and each point on the LDGM-BC curve corresponds to a code drawn from the irregular ensemble with a particular latency. We observe that the *regular* SC-LDGM codes outperform the irregular LDGM-BC as long as W is not too small and that performance tradeoffs can be achieved for SC-LDGM codes by varying M and W. In particular, increasing W improves the performance of the encoder, while increasing M improves the performance of the code.

2) Windowed encoding results for $N > 1$: Instead of decimating one code node at each iteration, we now set Algorithm B to decimate the $N > 1$ code nodes with the largest bias values. All numerical results presented are obtained by averaging over 1000 trials using an SC-LDGM code SC(4, 8) with coupling length $L = 55$, lifting factor $M = 512$, and window size $W = 8$, so the latency $2MW = 8192$ symbols and the compression rate $R_L = 0.5091$. For reference, the average distortion using Algorithm A (block encoding) is $\bar{D}_A = 0.112767$ after performing 28672 iterations.

We measure the *complexity* of the algorithm as the average number of updates $\bar{I}$ that a code node receives, averaged over the entire graph. Since these codes will typically operate in a regime with moderate to large L, we also present the average number of updates per node for nodes *outside* the initial window position, referred to as the average "*steady-state*" number of updates $\bar{I}_{ss}$, which is constant and larger than $\bar{I}$, since the nodes in the initial window position typically (depending on I_f) require fewer updates. We note that both $\bar{I}$ and $\bar{I}_{ss}$ depend on W, but, while $\bar{I}_{ss}$ is independent of L, $\bar{I}$ depends (slightly) on L. Consequently, we use $\bar{I}_{ss}$ to compare with Algorithm A, for which the average number of updates per code node is given by $\bar{I}_A = \frac{m(m+1)}{2m} = \frac{m+1}{2}$, where m is the number of code nodes. In this example, $\bar{I}_A = 14337$.

TABLE II
AVERAGE DISTORTION AND COMPLEXITY RESULTS FOR ALGORITHM B COMPARED TO ALGORITHM A.

N	$\bar{D}$	Distortion Loss (%)	$\bar{I}$	$\bar{I}_{ss}$	Complexity savings (%)
1	0.116905	3.6704	3975.80	4352.40	69.6422
2	0.117226	3.9542	1991.80	2176.50	84.8185
3	0.117680	4.3568	1332.30	1453.80	89.8594
4	0.118159	4.7815	999.75	1088.50	92.4075
8	0.119842	6.2740	503.75	544.50	96.2020
10	0.120622	6.9657	409.81	442.11	96.9162
20	0.122804	8.9007	208.51	221.28	98.4565
40	0.125296	11.1105	107.86	110.89	99.2265

Table II shows the performance of Algorithm B compared to Algorithm A, in terms of the *average distortion loss* (measured as $\frac{\bar{D}-\bar{D}_A}{\bar{D}_A}$) and the *average complexity savings* (measured as $\frac{\bar{I}_A-\bar{I}_{ss}}{\bar{I}_A}$), both given as percentage values, for the SC(4, 8) example with $I_f = 10$ and various values of N. We note that by increasing the number N of decimated code nodes at each iteration the complexity decreases accordingly; however, the distortion increases. For example, by decimating $N = 2$ code nodes per iteration, the algorithm complexity reduces to 84.82% at a cost of only 3.95% in distortion, whereas decimating $N = 10$ nodes per iteration results in a complexity reduction of 96.92% at a cost of 6.97% in distortion. By further increasing N to 20 (resp. 40) we obtain an average complexity savings of 98.46% (resp. 99.23%), but this costs 8.90% (resp. 11.11%) in distortion. We also note that, for $I_f = 10$, the savings in total complexity $\bar{I}$ is even larger, and that the complexity savings grows with increasing L. We will see in the next subsection that such reductions in complexity can be achieved with much smaller losses in distortion by implementing a thresholding scheme.

B. Thresholding (Algorithm C)

Instead of decimating a fixed number of nodes per iteration, the thresholding algorithm decimates all code nodes that have bias values greater than a pre-selected threshold θ after I_h iterations. The details are given in Algorithm C. Similar to Algorithm B, it is necessary to update the nodes in the first window position a sufficient number of times (I_f iterations) in order to generate bias values large enough to propagate through the chain. To insure termination, if no unassigned target node achieves a bias value larger than θ for 10 consecutive iterations, all remaining target nodes are decimated and the window shifts to the next position. Also, the parameter I_h, which can be used to control the speed of decimation, is normally set to 1, *i.e.*, we decimate after each iteration.[3] (Unless otherwise noted, we set $I_h = 1$ in the remainder of the paper.)

1) Numerical results and encoder dynamics of thresholding: Table III shows the performance of Algorithm C compared to Algorithm A for various threshold values θ and $I_f = 100$. We observe that the thresholding scheme provides a significant complexity savings compared to the benchmark case (more than 98% for all threshold values). Also, as expected, we observe that large threshold values θ provide good average distortion $\bar{D}$ but require more iterations (large $\bar{I}_{ss}$), whereas decreasing θ results in fewer iterations at a cost of larger distortion. We further note that decimation with thresholding can achieve similar reductions in complexity as decimating the best N nodes, but with better distortion. For example, both $N = 40$ (Algorithm B) and $\theta = 0.90$ (Algorithm C) have $\bar{I}_{ss} \approx 110$, but with average distortion losses of 11.11% and 6.19%, respectively. The improvement is also significant when comparing $N = 20$, with $\bar{I}_{ss} \approx 220$, to $\theta = 0.9999$, with $\bar{I}_{ss} \approx 200$, resulting in distortion losses of 8.90% and 4.19%, respectively. Moreover, it is simpler to compare each remaining target bias value to a threshold θ (Alg. C) than to find the N largest bias values (Alg. B); however, Alg. B provides a fixed complexity and run-time, whereas the complexity of Alg. C can vary.

Fig. 11 shows the cumulative number of decimated code nodes at each iteration for various θ. We observe that the algorithm has three phases:

1) the *initialization phase*, where we perform I_f iterations without hard decimation;
2) the *steady-state phase*, where the window moves at a constant speed over the graph (seen as an approximate straight line with slope Δ in Fig. 11); and
3) the *termination phase*, where the window reaches the end of the graph and the number of decimated nodes per iteration decreases to zero.

In particular, we see that the slope Δ in the steady-state phase

[3] In certain cases, *e.g.*, for large windows or high degree codes, thresholding can generate large bias values quickly, which can cause the algorithm to decimate too fast. In this case, complexity is low but the distortion can be unacceptably large. This can be controlled by increasing I_h.

Algorithm C: Threshold Decimation

```
1) At iteration $t = 0$, initialize the graph to $\Gamma^{(t=0)}$ and set a $counter \leftarrow 10$
2) $\forall i \in W_Z^{(t)}, a \in W_G^{(t)}$ update (4), (5), and (6) for $I_f - 1$ iterations such that
    a) there is no incoming or outgoing message $\forall a \in W_{G_f}^{(t)}$
    b) there are incoming messages $\forall i \in W_{Z_p}^{(t)}$ and $\forall a \in W_{G_p}^{(t)}$, but no outgoing messages to those nodes
3) $\forall i \in W_Z^{(t)}, a \in W_G^{(t)}$ update (4), (5), and (6) for $I_h$ iterations, under the same conditions as 2)
4) For each remaining bias value $B_i^{(t)}$, $i$ not fixed, $i \in T(W^{(t)})$, do
   **If** $|B_i^{(t)}| \geq \theta$ then
    a) **if** $B_i^{(t)} > 0$ then
         $z_i \leftarrow '1'$
       **else**
         $z_i \leftarrow '0'$
       **end if**
    b) decimate graph as
        i) $\forall a \in G(i)$, $s_a \leftarrow s_a \oplus z_i$ (update source symbols)
        ii) reduce the graph as $\Gamma^{(t+1)} = \Gamma^{(t)} \backslash \{i\}$ (remove code node $i$ and all its edges)
    c) $counter \leftarrow 10$
   **end if**
5) **If** no nodes were decimated in Step 4) then
      $counter \leftarrow counter - 1$
      **If** $counter = 0$
         For $\{B_i^{(t)} | \forall i$ not fixed, $i \in T(W^{(t)})\}$, decimate $B_i^{(t)}$ according to 4a) and 4b)
         $counter \leftarrow 10$
      **else** go to 3)
      **end if**
    **end if**
6) **if** there exist any unassigned code symbols $i \in T(W^{(t)})$, go to 3)
   **else if** all source symbols are not yet encoded, shift window to the next position, go to 3)
   **else** exit algorithm
   **end if**
```

TABLE III
AVERAGE DISTORTION AND COMPLEXITY RESULTS FOR ALGORITHM C COMPARED TO ALGORITHM A.

Threshold value θ	$\bar{D}$	Distortion Loss (%)	$\bar{I}_{ss}$	Complexity savings (%)
0.9999	0.117495	4.1927	202.37	98.5884
0.999	0.117977	4.6201	167.21	98.8337
0.99	0.118411	5.0050	137.04	99.0441
0.90	0.119747	6.1898	112.40	99.2160
0.80	0.120408	6.7759	105.71	99.2627

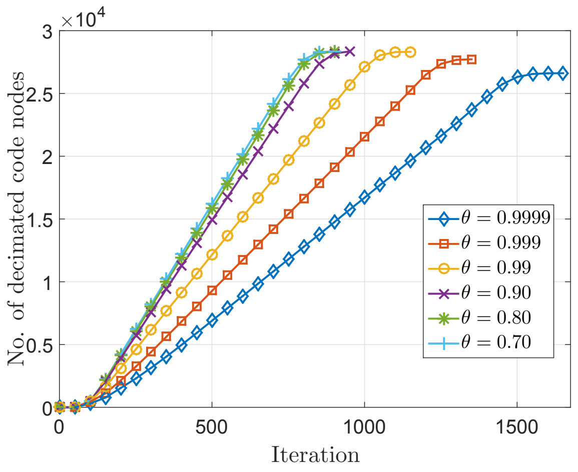


Fig. 11. Cumulative number of decimated code nodes at each iteration for different threshold values θ.

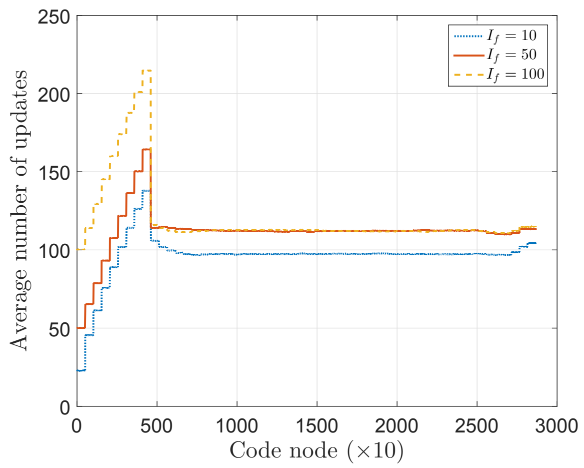


Fig. 12. Average number of updates per code node for various I_f.

TABLE IV
Performance of Algorithm C in the steady-state phase.

Threshold value θ	Δ	$\bar{D}_{ss}$	$\bar{I}_{ss}$
0.9999	20.3918	0.1181	202.3663
0.999	24.6209	0.1184	167.2043
0.99	30.1674	0.1191	137.0379
0.90	36.7937	0.1204	112.3921
0.80	39.2504	0.1214	105.7014
0.70	40.1335	0.1216	103.1221

TABLE V
Average distortion results for $\theta = 0.90$ and various I_f.

I_f	$\bar{D}$	$\bar{D}_{ss}$	$\bar{I}_{ss}$
500	0.1197	0.1204	112.6404
100	0.1197	0.1204	112.3921
50	0.1202	0.1205	112.1783
10	0.1232	0.1236	97.4348

indicates that the algorithm decimates approximately Δ code nodes per iteration as the window moves over the graph and that Δ is larger for lower thresholds θ. The values of Δ are shown in Table IV, where we note that the decimation rate roughly doubles from $\theta = 0.9999$ to $\theta = 0.70$ and, assuming that a large L will be used in practice, the performance is dominated by the steady-state phase. Also given in Table IV are the values obtained for the *average steady-state distortion* $\bar{D}_{ss}$, obtained as the average distortion over all nodes excluding those at the ends (initialization and termination phases), along with the average number of steady-state node updates $\bar{I}_{ss}$. The results indicate that $\bar{D}_{ss}$ approaches the overall average distortion $\bar{D}$ for large θ, with relatively low complexity, which in turn approaches the RD limit for (4, 8)-regular LDGM codes [27] with increasing M. We also see that relaxing the threshold results in higher steady-state distortion but lower complexity.

Due to the initialization phase, the complexity is also affected by I_f. Fig. 12 shows the complexity, measured as the average number of updates per code node, as a function of position in the graph, for $\theta = 0.90$ and $I_f = 10$, 50, and 100. Again, we observe that the encoding can be divided into three phases: initial, steady-state, and termination. Since $W = 8$ and $M = 512$, the first 9 sections ($9M = 4608$ code nodes) automatically receive at least I_f updates (cf. Fig. 2). This is evident as the "steps" in Fig. 12. As the algorithm progresses, we reach the steady-state phase where the complexity and the decimation rate Δ are constant (see also Fig. 11). We note that the number of updates in the latter positions of the initial window exceeds the steady-state values, due to the offset of I_f, but that this behavior is less pronounced for small values of I_f. Finally, we reach the termination phase, where Fig. 12 shows a slight increase in complexity. Table V summarizes the distortion results for $\theta = 0.90$ and various values of I_f. We observe that, by increasing I_f, the distortion decreases and the complexity increases, and we note that it is important to have a sufficiently large I_f to generate reliable bias values, but increasing I_f further offers diminishing gains.

2) *SC-LDGM thresholding performance for different compression rates:* Although we have focused on SC-LDGM code ensembles with nominal compression rate $R = 0.5$, similar performance can be observed for a variety of compression rates, with and without WE. Fig. 13 summarizes simulation results of the average steady-state distortion $\bar{D}_{ss}$ for various compression rates R compared to the corresponding RD limit. (J, K)-regular SC-LDGM codes with fixed generator node degree J were constructed, which we denote by LDGM(J). The constructed codes have lifting factor $M = 512$, which results in an overall LDGM code length of approximately 100000. They were encoded using Algorithm C, with $\theta = 0.9999$, window size $W = 7$, ξ optimized to three decimal places, $I_f = 100$, and $I_h = 14$ iterations between two successive decimation steps.[4]

As can be seen from Fig. 13, the distortion is very close to the RD limit in all cases. We also observe that by increasing the rate, the average distortion gap to the RD limit is

[4] $I_h = 14$ was chosen here since it results in good distortion for all cases considered in Fig. 13. This value can be reduced to $I_h = 1$ in many cases without significant loss in distortion, as seen earlier for some $(J, 2J)$-regular ensembles.

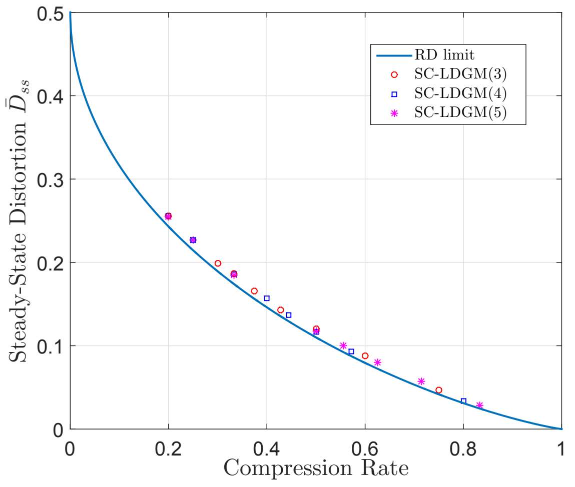


Fig. 13. Average steady-state distortion of SC-LDGM codes for three different generator node degrees J and various compression rates.

diminishing. Furthermore, we note that the distortion of the LDGM(J) ensembles approaches the RD limit as the generator node degree J increases. These results were not optimized on a case-by-case basis (with the exception of ξ^*); indeed, the simulation parameters were simply chosen such that we expected "good" performance. Consequently, they can likely be improved in terms of providing a better latency/complexity trade-off for a given distortion by optimizing parameters, such as the protograph structure, further increasing the accuracy of ξ^*, and adjusting the threshold θ, the number of initial iterations I_f, the number of iterations between decimation steps I_h, and the window size W.

C. A Complexity and Latency Comparison with LDGM-BCs

This section briefly compares the proposed protograph-based SC-LDGM codes with an example of a state-of-the-art, highly irregular LDGM-BC that was optimized for the soft-decimation algorithm [25]. We showed in Section IV-A that, on an equal latency basis, the *regular* SC(4, 8) SC-LDGM code encoded with WE outperformed an optimized *irregular* code. For the purpose of illustration, we now fix the latency to be 8192 symbols and allow the complexity to vary for comparison. The irregular LDGM-BC is constructed with degree distribution as given in (8) and (9) following [25] and the SC(4, 8) code is constructed with $M = 512$ and encoded with $W = 8$. Comparing Tables IV (for the SC-LDGM code) and VI (for the LDGM-BC), we see that, for equal rate and fixed latency, the average steady-state distortion $\bar{D}_{ss}$ of the regular SC-LDGM code is less than the average distortion $\bar{D}$ of the optimized irregular LDGM-BC when the average number of steady-state node updates $\bar{I}_{ss}$ of the SC-LDGM code equals the number of iterations of the LDGM-BC. Further gains for the SC-LDGM code may be possible by optimizing the code construction and the windowed encoder parameters.

D. Min-Sum Algorithm

Following well-established procedures from the channel coding literature to achieve good performance-complexity

TABLE VI
SOFT DECIMATION [25] RESULTS FOR AN IRREGULAR LDGM-BC WITH $n = 8192$ AND RATE $R_L = 0.5$ (RD LIMIT 0.11).

Algorithm iterations	$\bar{D}$
202	0.119420
167	0.120095
137	0.120855
112	0.121955
105	0.122237
103	0.122309

TABLE VII
EFFECT OF THE LIFTING FACTOR M FOR THE MIN-SUM ALGORITHM ON SC(4, 8) CODES IN TERMS OF AVERAGE DISTORTION $\bar{D}$ AND AVERAGE GAP TO THE RD LIMIT $\bar{\delta}$ WITH FIXED $L = 16$ AND COMPRESSION RATE $R_L = 0.5313$.

Lifting factor (M)	$\bar{D}$	$\bar{\delta}$	Deviation from BP Alg. (%)
256	0.111921	0.0120	1.6780
640	0.110373	0.0105	1.5905
832	0.110035	0.0101	1.4353
1024	0.109860	0.0100	1.3506
2048	0.109314	0.0094	1.0595

trade-offs, the proposed decimation algorithm can be simplified by approximating the generator node to code node message using a simple minimum operation, known as the min-sum algorithm. For this purpose, (5) is replaced with

$$R^{(t+1)}_{a\to i} = \frac{1}{\mu} R^{(t)}_{i\to a} + \gamma(-1)^{s_a+|Z(a)|} \left(\prod_{j\in Z(a)\setminus i} \alpha^{(t)}_{j\to a} \right) \min\{\psi\}; \tag{10}$$

where $\gamma < 1$ is a constant scaling factor, $\alpha^{(t)}_{j\to a} = sign\left(R^{(t)}_{j\to a}\right)$, and $\psi = \{|R^{(t)}_{j\to a}|\}_{j\in Z(a)\setminus i}$ (see [34] for more details.)

To obtain results for the min-sum algorithm, we limited the magnitudes of all messages (code to generator node and generator to code node) to 20 and used $\gamma = 0.5$.[5] Then, replacing (5) with (10) in Alg. A (block encoding) for an SC(4, 8) code with coupling length $L = 55$ and lifting factor $M = 512$, we obtained distortion $\bar{D} = 0.114440$ (compared to the BP version of Alg. A with $\bar{D}_A = 0.112767$). In addition, we investigated the effect of the lifting factor M using the min-sum algorithm for some of the codes given in Table I. The obtained distortions are shown in the Table VII, where we see that the performance degradation is very slight (less than 1.68%) and decreases with increasing lifting factor M for the min-sum case.

V. Conclusion

In this paper, we introduced a new construction of (J, K) regular SC-LDGM codes based on protographs for lossy source coding. The proposed "soft-hard" BP-based decimation algorithm was shown to perform well, giving average distortion values close to the RD limit. We then presented a novel low-latency WE algorithm with two decimation approaches

[5]Several approaches could be taken here to improve distortion performance, such as optimizing the pair (ξ, γ), allowing γ to vary over iterations, or applying a normalized version of the min-sum algorithm.

that can significantly lower complexity (measured as the number of node updates) with little performance degradation for a sufficiently large window size, allowing good source reconstruction performance to be obtained for moderate encoding latency. The first technique decimates a fixed number of nodes per iteration and the second decimates all nodes with sufficiently large confidence at each iteration. We showed that a significant reduction in complexity ($> 90\%$) can be achieved compared to decimating a single node per iteration, with only moderate losses in distortion. Also, we showed that a min-sum formulation can be used during the BP updates that further reduces the complexity at the cost of a slight loss in performance. There are several features of WE approach that can be improved, such as designing good convolutional protographs that permit shorter window sizes and optimizing message passing schedules within an encoding window. This, along with a study of the performance for more general sources, is the subject of ongoing work.

Acknowledgements

The authors would like to thank the reviewers for their constructive comments to improve the presentation of the paper. In particular, the comments of Reviewer 1 regarding the introduction of a min-sum formulation are especially appreciated.

References

[1] T. Goblick, "Coding for a discrete information source with a distortion measure," *Ph.D. dissertation*, Massachussets Institute of Technology, Cambridge, MA, 1962.

[2] A. Viterbi and J. Omura, "Trellis encoding of memoryless discrete-time sources with a fidelity criterion," *IEEE Trans. Inf. Theory*, vol. 20, no. 3, pp. 325-332, Mar. 1974.

[3] M. Marcellin and T. Fischer, "Trellis coded quantization of memoryless and Gauss-Markov sources," *IEEE Transactions on Communications*, vol. 38, no. 1, pp. 82-93, Jan. 1990.

[4] J. Garcia-Frias and Y. Zhao, "Compression of binary memoryless sources using punctured turbo codes," *IEEE Communications Letters*, vol. 6, no. 9, pp. 394-396, 2002.

[5] Y. Matsunaga and H. Yamamoto, "A coding theorem for lossy data compression by LDPC codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 9, pp. 2225-2229, Sept. 2003.

[6] N. Hussami, S. B. Korada, and R. Urbanke, "Performance of polar codes for channel and source coding," in *Proc. IEEE International Symposium on Information Theory*, pp. 1488-1492, 2009.

[7] M. Karzand, and E. Telatar, "Polar codes for q-ary source coding," in *Proc. IEEE International Symposium on Information Theory*, pp. 909-912, 2010.

[8] S. B. Korada and R. L. Urbanke, "Polar codes are optimal for lossy source coding," *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1751-1768, 2010.

[9] M. Wainwright and E. Martinian, "Low-density graph codes that are optimal for source/channel coding and binning," *IEEE Transactions on Information Theory*, vol. 55, no. 3, 2009.

[10] M. J. Wainwright, E. Maneva, and E. Martinian. "Lossy source compression using low-density generator matrix codes: Analysis and algorithms," *IEEE Trans. Inf. Theory*, vol. 56, no. 3, pp. 1351-1368, 2010.

[11] A. Dimakis, M. Wainwright, and K. Ramchandran, "Lower bounds on the rate-distortion function of LDGM codes," in *Proc. IEEE Inf. Theory Workshop*, pp. 650-655, 2007.

[12] S. Kudekar and R. Urbanke, "Lower bounds on the rate-distortion function of individual LDGM codes," in *Proc. Int. Symp. on Turbo Codes and Related Topics*, pp. 379-384, 2008.

[13] F. Krzakala, A. Montanari, F. Ricci-Tersenghi, G. Semerjian, and L. Zdeborova, "Gibbs states and the set of solutions of random constraint satisfaction problems," *Proc. of the National Academy of Sciences of the United States of America*, vol. 105, no. 25, pp. 10 318-10 323, 2007.

[14] N. Noorshams and M. Wainwright, "Lossy source coding with sparse graph codes: A variational formulation of soft decimation," *in Proc. 48th Annual Allerton Conference on Communication, Control, and Computing*, pp. 354-360, 2010.

[15] T. Murayama, "Thouless-Anderson-Palmer approach for lossy compression," *Physical Review E*, vol. 69, no. 035105, pp. 1-4, 2004.

[16] M. J. Wainwright and E. Maneva. "Lossy source encoding via message-passing and decimation over generalized codewords of LDGM codes," in *Proc. IEEE International Symposium on Information Theory*, Adelaide, Australia, September 2005.

[17] S. Ciliberti, M. Mézard, and R. Zecchina, "Message-passing algorithms for non-linear codes and data compression," *Complexus*, vol. 3, no. 58, pp. 58-65, Aug. 2006.

[18] S. Ciliberti and M. Mézard, "The theoretical capacity of the parity source coder," *J. Stat. Mech.*, vol. 10, no. P10003, pp. 1-14, 2006.

[19] A. Montanari, F. Ricci-Tersenghi, and G. Semerjian, "Solving constraint satisfaction problems through belief propagation-guided decimation," *in Proc. 45th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Oct. 2007.

[20] A. Coja-Oghlan, "A better algorithm for random k-sat," *SIAM Journal on Computing*, vol. 39, no. 7, pp. 2823-2864, 2010.

[21] E. Martinian and J. S. Yedidia, "Iterative quantization using codes on graphs," in *Proc. Allerton Conference on Comm., Control, and Computing*, Monticello, IL, Oct. 2003.

[22] G. Demay, V. Rathi, and L. K. Rasmussen, "Optimality of LDGM-LDPC compound codes for lossy compression of binary erasure source," *in Proc. International Symposium on Information Theory and its Applications (ISITA)*, pp. 589-594, 2010.

[23] G. Demay, V. Rathi, and L. K. Rasmussen, "Rate distortion bounds for binary erasure source using sparse graph codes," *in Data Compression Conference*, pp. 49-58, 2010.

[24] T. Filler and J. Fridrich, "Binary quantization using belief propagation with decimation over factor graphs of LDGM codes," in *Proc. 45th Annual Allerton Conf. on Comm., Control, and Computing*, Monticello, IL, Oct. 2007.

[25] D. Castanheira and A. Gameiro, "Lossy source coding using belief propagation and soft-decimation over LDGM codes," in *Proc. IEEE Int. Symp. Personal Indoor and Mobile Radio Communications*, Istanbul, Turkey, Sept. 2010.

[26] V. Aref, N. Macris, M. Vuffray, "Approaching the Rate-Distortion Limit With Spatial Coupling, Belief Propagation, and Decimation," in *IEEE Trans. Inf. Theory*, vol. 61, no. 7, pp. 3954-3979, July 2015.

[27] V. Aref, N. Macris, R. Urbanke, M. Vuffray, "Lossy source coding via spatially coupled LDGM ensembles," in *Proc. IEEE Int. Symp. Inf. Theory*, pp. 373-377, July 2012.

[28] D. Divsalar, S. Dolinar, C. Jones, and K. Andrews, "Capacity-approaching protograph codes," *IEEE J. Sel. Areas in Comm.*, vol. 27, no. 6, pp. 876-888, Aug. 2009.

[29] T. Berger and J. D. Gibson, "Lossy source coding," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2693-2723, 1998.

[30] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley and Sons, 2012.

[31] J. Thorpe, "Low-density parity-check LDPC codes constructed from protographs," *IPN Progress Report, Tech.* Rep. 42-154, Aug. 2003.

[32] D. G. M. Mitchell, M. Lentmaier, D. J. Costello, "Spatially Coupled LDPC Codes Constructed From Protographs," *IEEE Trans. Inf. Theory*, vol. 61, no. 9, pp. 4866-4889, Sept. 2015.

[33] A. R. Iyengar, M. Papaleo, P. H. Siegel, J. K. Wolf, A. Vanelli-Coralli, and G. E. Corazza, "Windowed decoding of protograph-based LDPC convolutional codes over erasure channels." *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2303-2320, 2012.

[34] W. E. Ryan and S. Lin. *Channel codes: classical and modern*. New York, NY, USA: Cambridge University Press, 2009.