On the Windowed Encoding Complexity of SC-LDGM Codes for Lossy Source Compression

Ahmad Golmohammadi*, Jörg Kliewer†, Daniel J. Costello, Jr.‡, and David G. M. Mitchell*
*Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, NM, USA, {golmoham, dgmm}@nmsu.edu
†Dept. of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA, jkliewer@njit.edu
‡Dept. of Electrical Engineering, University of Notre Dame, Notre Dame, IN, USA, costello.2@nd.edu

***Abstract*—It has been shown that spatially coupled low-density generator-matrix (SC-LDGM) code ensembles display distortion saturation for the lossy binary symmetric source coding problem with belief propagation guided decimation algorithms, in the sense that the distortion of the SC-LDGM code ensemble approaches the optimal distortion of the underlying (uncoupled) LDGM block code ensemble. This has also been demonstrated for the class of protograph-based SC-LDGM code ensembles with windowed encoding (WE), where distortion close to the rate distortion limit was obtained with low-latency encoding. In this paper, we propose and compare two decimation techniques for lowering the WE complexity of SC-LDGM codes that maintain distortion performance close to the rate-distortion bound.**

I. Introduction

Like low-density parity-check block codes (LDPC-BCs), low-density generator-matrix block codes (LDGM-BCs) can be defined on a sparse graph and are amenable to low complexity message passing algorithms. Unfortunately, directly applying such an approach to the lossy source coding problem typically fails because there are multiple optimal (or near-optimal) solutions and one cannot find the relevant fixed point without reducing the solution space. Indeed, the sum product algorithm (SPA), which is known to achieve excellent performance for channel coding problems, does not give satisfactory results in the lossy source coding case because the approximate marginal calculated by the SPA does not provide reliable information about optimal encoding [1]. To combat this, modified belief propagation (BP) algorithms which include a *decimation* step, including both hard-decimation [1] and soft-decimation [2] algorithms, have been proposed. Regarding code design, the best known LDGM-BC designs are typically dual codes to LDPC-BCs that were optimized for the binary symmetric channel (BSC) [1]–[4].

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 [5], [6], a belief propagation guided decimation (BPGD) algorithm was applied to a class of SC-LDGM codes with regular check node degrees and Poisson distributed variable node degrees for lossy source compression of the binary symmetric source. 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 L and code length tend to infinity, which, in turn, approaches the rate distortion (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.

In [7], distortion saturation was demonstrated numerically for a class of practically interesting protograph-based SC-LDGM codes and a windowed encoding (WE) algorithm was presented that maintains performance close to the RD limit but with drastically reduced latency and memory requirements. However, the algorithm presented in that paper decimates a single code node per iteration, which results in good performance but high computational complexity. In this paper, we present two decimation approaches for WE that can significantly lower 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. Finally, we discuss the latency, complexity, and performance trade-offs of WE and compare our results with an optimized irregular LDGM-BC.

II. SC-LDGM Codes for lossy compression

A. Lossy source compression

We consider compressing the *symmetric Bernoulli source*, 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 $\mathbb{P}(s_i = 1) = 1/2$. We wish to represent a given source sequence by a 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 $D = \mathbb{E}_{\mathbf{s}}[d_H(\mathbf{s}, \hat{\mathbf{s}})]$. For any encoding scheme, the average distortion is bounded below by D_{Sh}, defined implicitly by the RD function $h(D_{Sh}) = 1 - R, D_{Sh} \in [0, 0.5]$, where $h(\cdot)$ is the binary entropy function.

B. Protograph-based SC-LDGM codes

An SC-LDGM code or code ensemble with coupling length L can be represented by means of a *convolutional protograph* [7]. The protograph is 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, and it can be represented by a generator or *base* biadjacency matrix

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

where w denotes the *coupling width*, the $b_c \times b_r$ matrices $\mathbf{B}_i$ are referred to as *component base matrices*, $i = 0, 1, \ldots, w$, and $B_{x,y}$ is taken to be the number of edges connecting generator node g_y to code node z_x. The generator matrix $\mathbf{G}_{[0,L-1]}$ of a protograph-based SC-LDGM code can be created by the *copy-and-permute* operation [8], replacing each non-zero entry in $\mathbf{B}_{[0,L-1]}$ by a sum of $B_{x,y}$ non-overlapping permutation matrices of size $M \times M$ and each zero entry by the $M \times M$ all-zero matrix. An ensemble of SC-LDGM codes can then be formed from $\mathbf{B}_{[0,L-1]}$ using this copy-and-permute method over all possible permutation matrices. The *design compression rate* of the ensemble of SC-LDGM 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 $L \to \infty$.

In this paper, we use $(4, 8)$-regular SC-LDGM code ensembles as a representative example to demonstrate our results, where the component base matrices are chosen as $\mathbf{B}_0 = \mathbf{B}_1 = \mathbf{B}_2 = \mathbf{B}_3 = [1, 1]$. In order to improve distortion performance for BP algorithms, analogous to [5], [7] we reduce the generator node degrees at the start of the protograph by deleting the first 2 rows of (1). The resulting ensembles are denoted SC$(4, 8)$ and have modified design compression rate

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

Fig. 1 depicts the protograph representation of the SC$(4, 8)$ SC-LDGM code ensemble, where 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. See [7] for other constructions and further details of the protograph construction.

III. Source Reconstruction Using a Windowed Encoding Scheme

A. Decimation Algorithm

In this paper, we use a mixture of hard and soft decimation following [7]. We apply the following BP update equations

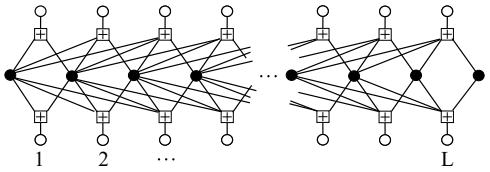

Fig. 1: Protograph of the (4,8)-regular SC-LDGM ensemble SC$(4, 8)$.

at each iteration t [2]:

Code to generator node messages:

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

Generator to code node messages:

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

Code node bias update:

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

where $R_{i\to a}^{(t)}$, $R_{a\to i}^{(t)}$, and $B_{i\to a}^{(t)}$ 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 $R_{i\to a}^{(t)}$ at iteration t, respectively, and β and μ are parameters. Then $R_i^{(t)}$ and $B_i^{(t)}$ denote the likelihood ratio of code node i and the bias associated with $R_i^{(t)}$, respectively. 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 and $G(i)$ denotes the set of all generator node indices connected to code node i, $\forall i \in Z$ and $\forall a \in G$. The decimation process is embedded in (5) by the $\frac{1}{\mu} R_{i\to a}^{(t)}$ term, which scales the belief of the code nodes at each iteration [2]. Here, the parameter $\mu > 0$ determines the accuracy of the approximation. Since β and μ are free parameters, they can be combined as $\beta = (1-\xi)/(1+\xi)$ and $\mu = 1/\xi$ for simplicity. This choice was shown to yield good numerical results in [2].

B. Windowed encoding

For practical implementations of SC-LDGM codes, it is essential to reduce the encoding latency. To this end, we apply the sliding window encoder proposed in [7], where a window of size W (containing W sections of the graph) slides over the graph continuously. At each window position, the modified BP algorithm is applied to a fraction of the code nodes and all their neighboring generator nodes in order to encode a subset of code symbols, called *target* symbols. After encoding the current target symbols (i.e., when they are all decimated), the window slides over one section of the graph and again executes the modified BP algorithm to encode the next set of target symbols, using both the current source symbols and some previously encoded source symbols. Fig. 2 shows the WE procedure on the protograph of a SC$(4, 8)$ code with window size $W = 3$ (covering 3 graph sections, or $3M$ code symbols). Here $2M$ source symbols enter the window at each

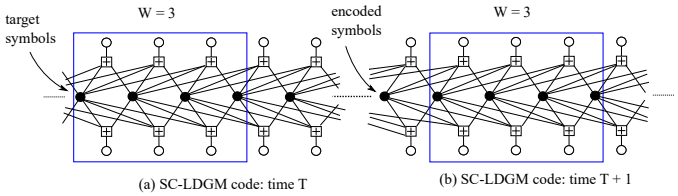

Fig. 2: The WE procedure operating on the protograph of an SC$(4, 8)$ code.

window position and M source symbols leave (are encoded). See [7] for full details.

IV. Complexity of Windowed Encoding

In [7], it was shown that the WE algorithm applied to SC(J, K) codes provides distortion values that saturate to a value close to the RD limit for a sufficiently large coupling length L and window size W. In that paper, the algorithm decimates one code node per iteration, and consequently the algorithm performs a large number of iterations. To decrease the complexity of WE, we propose two modifications of the algorithm in [7]. All numerical results presented are obtained by averaging results over 1000 trials using an SC-LDGM code SC$(4, 8)$ with coupling length $L = 55$, lifting factor $M = 512$, and window size $W = 8$. This results in latency $2MW = 8192$ and code rate $R_L = 0.5091$. For each simulation, ξ was determined experimentally to three decimal places to give the lowest distortion value. For reference, the obtained distortion from the algorithm in [7] for $W = 8$ is $D = 0.116906$ after performing $I_{tot} = 28672$ iterations. Results for the proposed schemes will be given as a percentage deficiency from these values.

1) (Algorithm A) Multiple node decimations per iteration: Instead of decimating one code node at each iteration, this algorithm decimates the $N \geq 1$ code nodes with the highest bias values.[1] 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, before decimation begins.[2] Details are given in Algorithm A, where t indicates the iteration number, $\Gamma^{(t)}$ is the LDGM code graph at iteration t, i is the code node location, z_i represents the binary value assigned to the code node i, $W^{(t)}$ 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 $T\left(W^{(t)}\right)$ denotes the 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 $+0.1$ and reset at iteration 1.

Table I shows the distortion deficiency for the SC$(4, 8)$ example with $I_f = 10$ and various values of N. The cumulative number of iterations I_{tot} over all window positions is used to obtain a rough percentage speed up over the benchmark case. Since each node (except those in the first 7 sections)

[1]Searching for the largest N bias value(s) has an additional computational cost, this will be avoided by the thresholding approach in Algorithm B.

[2]These reliable bias values then propagate through the graph from left to right as we iterate following the "wave-like encoding" model (see [7]).

Algorithm A: Multiple node decimation scheme

1) At iteration $t = 0$, initialize the graph to $\Gamma^{(t=0)}$
2) Update (4), (5), and (6) for $I_f - 1$ iterations
3) $\forall i \in W_Z^{(t)}, a \in W_G^{(t)}$ update (4), (5), and (6) 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
4) Find the N largest bias values among the target symbols $B^{(t)} = \max_i\{B_i^{(t)} |\ i \text{ not fixed}, i \in T(W^{(t)})\}$
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 symbol $i \in T(W^{(t)})$, go to 3)
   else if all source symbols are not encoded, shift window to the next position, go to 3)
   else exit algorithm and return code symbols
   end if

participates in 8 windows, we also present the average number of iterations for nodes outside the initial window position $\bar{I}_{ss}$ (the *average* "*steady-state*" *iterations*), as a measure of algorithm complexity. We see that by increasing the number

TABLE I: Performance and complexity results for Algorithm A.

N	Distortion D	Distortion Deficiency (%)	I_{tot}	Speed up(%)	$\bar{I}_{ss}$
2	0.117226	0.2737	14345	49.9686	2176.5000
3	0.117680	0.6621	9585	66.5702	1453.8000
4	0.118159	1.0718	7177	74.9686	1088.5000
8	0.119842	2.5114	3593	87.4686	544.5041
10	0.120622	3.1786	2921	89.8124	442.1025
20	0.122804	5.0451	1465	94.8905	221.2770
40	0.125296	7.1767	737	97.4295	110.8872

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 speeds up by almost 50% at a cost of only 0.27% in distortion, whereas decimating $N = 10$ nodes per iteration results in a speed up of around 90% at a cost of 3.18% in distortion. By further increasing N to 20 (resp. 40) we obtain an average number of updates of 221.28 (110.89) respectively, but this costs 5.05% (7.18%) in distortion. We will see next that such a reduction in complexity can be achieved with a much smaller loss in performance by implementing a thresholding scheme.

2) (Algorithm B) Threshold decimation: Instead of decimating a fixed number of nodes per iteration, this algorithm decimates all code nodes that have bias greater than a selected threshold value θ at each iteration. Similar to Algorithm A,

Algorithm B: Threshold decimation scheme

1) At iteration $t = 0$, initialize the graph to $\Gamma^{(t=0)}$ and set a $counter \leftarrow 10$
2) Update (4), (5), and (6) for $I_f - 1$ iterations
3) $\forall i \in W_Z^{(t)}, a \in W_G^{(t)}$ update (4), (5), and (6) 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
4) Find the set of bias values $B^{(t)} = \{B_i^{(t)} \geq \theta \mid i \text{ not fixed}, i \in T(W^{(t)})\}$
5) **If** $\mid B^{(t)} \mid = 0$, then
   $counter \leftarrow counter - 1$
   If $counter = 0$
   $B^{(t)} = \{B_i^{(t)} \mid i \text{ not fixed}, i \in T(W^{(t)})\}$
   else go to 3)
   end if
   end if
6) For each of selected biases $i \in \{B^{(t)}\}$, 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)
7) **if** there exist any unassigned code symbol $i \in T(W^{(t)})$, go to 3)
   else if all source symbols are not encoded, shift window to the next position, go to 3)
   else exit algorithm and return code symbols
   end if

it is necessary to update the nodes in the first position a number of times (I_f iterations) in order for sufficient bias to be generated that can propagate through the chain. To force algorithm termination, if no code node achieves a bias larger than θ for 10 consecutive iterations, then all remaining target nodes in the current window are decimated and the window shifts to the next position. This algorithm is given in more detail as Algorithm B.

Table II shows the performance of Algorithm B for various threshold values θ. We observe that thresholding provides a significant speed up compared to the benchmark case (above 94% for all given threshold values). Also, we observe that large threshold values provide good distortion performance but require more iterations, whereas decreasing θ results in fewer iterations (both I_{tot} and $\bar{I}_{ss}$ are decreased) at a cost of a greater distortion deficiency. We note that decimation with thresholding can achieve similar reductions in complexity as decimating the best N nodes, but with less distortion deficiency. For example, both $N = 40$ (Algorithm A) and $\theta = 0.90$ (Algorithm B) have $\bar{I}_{ss} \approx 110$ but with distortion deficiencies of 7.18% and 2.43%, respectively. The improvement is equally significant comparing $N = 20$ and $\theta = 0.9999$, with $\bar{I}_{ss} \approx 220$ and 200, respectively, which result in distortion deficiencies of 5.05% and 0.50%. Moreover, it is simpler to compare each bias value to the threshold θ to determine the

TABLE II: Performance and complexity results for Algorithm B.

Threshold value θ	Distortion D	Distortion Deficiency (%)	I_{tot}	Speed up (%)	$\bar{I}_{ss}$
0.9999	0.117495	0.5038	1508.333	94.7394	202.3663
0.999	0.117977	0.9161	1272.721	95.5611	167.2043
0.99	0.118411	1.2874	1051.426	96.3329	137.0379
0.90	0.119747	2.4302	878.932	96.9345	112.3921
0.80	0.120408	2.9956	830.674	97.1028	105.7014
0.70	0.120657	3.2086	814.907	97.1578	103.1221

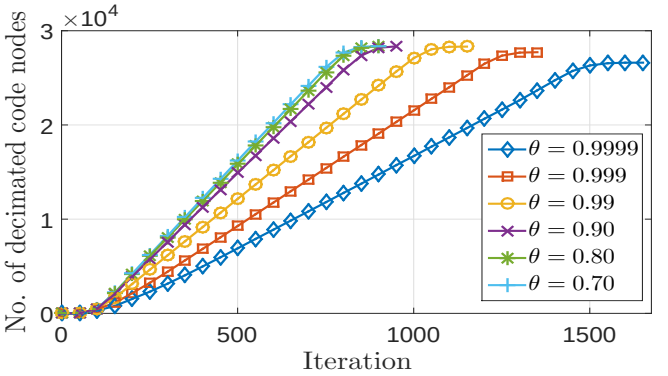

Fig. 3: Cumulative number of decimated code nodes at each iteration for different threshold values θ.

decimated nodes (Alg. B) than to find the N largest bias values (Alg. A); however, Alg. A provides a fixed complexity, whereas the complexity of Alg. B can vary.

Fig. 3 shows the number of cumulative 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. 3); and
3) the *termination phase*, where the window reaches the end of the graph and the number of decimated nodes per iteration decreases.

In particular, we see that the slope Δ in the steady-state phase indicates that the algorithm decimates approximately Δ code nodes on average at each iteration as the window moves over the graph, and Δ is larger for lower thresholds θ. The values of Δ are shown in Table III, where we see that the decimation rate roughly doubles from $\theta = 0.9999$ to $\theta = 0.70$.

We assume that a large L will be used in practice and thus the performance is dominated by the steady-state phase. Also given in Table III are the values obtained for the *steady-state distortion* D_{ss} and the average steady-state iterations $\bar{I}_{ss}$ for several values of θ with $I_f = 100$. The results indicate that the distortion approaches the referenced value for large θ, which, in turn, approaches the optimal RD limit for $(4, 8)$-regular LDGM codes [6] with increasing M, with relatively low complexity. We also see that relaxing the threshold results in higher steady-state distortion but lower complexity.

Fig. 4 shows the complexity measured as the average number of updates of each code node for three different values of I_f, 10, 50, and 100, with $\theta = 0.90$. Again, we observe that the encoding can be divided to three phases: initial, steady-

TABLE III: Performance of Algorithm B in the steady-state phase.

Threshold value θ	Δ	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

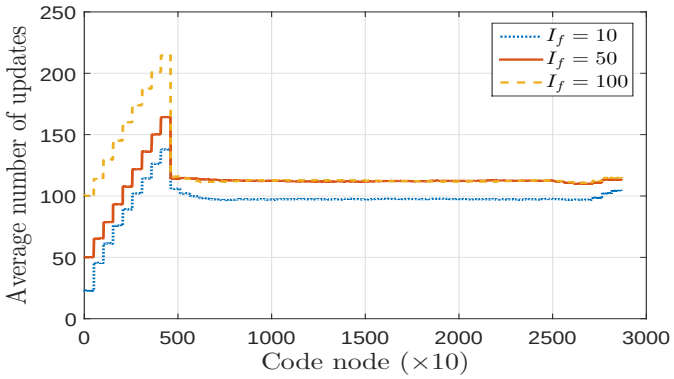

Fig. 4: Average number of updates per code node for various I_f.

state, and termination. Since $W = 8$, the first 9 sections ($9M$ code nodes) receive I_f updates and then, as the window shifts over the graph, receive an increasing number of updates. This is seen as the "steps" in Fig. 4. As the algorithm progresses, we reach the steady-state phase where the complexity and decimation rate Δ are constant (see Fig. 3). 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. 4 shows a slight increase in complexity for this value of θ. We see also from Fig. 3 that in this phase few code nodes are decimated. Considering the effect of I_f, Table IV shows the distortion and complexity with $\theta = 0.90$ for various I_f (see also Fig. 4). We observe that by increasing I_f, the distortion decreases and the complexity increases, where 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.

V. A Complexity and Latency Comparison with LDGM-BCs

It is beyond the scope of this paper to perform a full and detailed comparison of SC-LDGM codes with state-of-the-art LDGM-BCs, but we briefly discuss some key features in comparison to a highly irregular LDGM-BC that was optimized for the soft-decimation algorithm [2]. We showed in [7] that, on an equal latency basis, the *regular* SC$(4, 8)$ SC-LDGM code outperformed the optimized *irregular* code. Comparing Tables III and V, we now see that for equal rate, latency, and complexity, the regular SC-LDGM code outperforms the optimized irregular LDGM-BC. We expect further gains by optimizing the SC-LDGM code and the windowed encoder.

TABLE IV: Distortion results for $\theta = 0.90$ and various I_f.

I_f	D	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

TABLE V: Soft decimation [2] results for an *irregular* LDGM-BC with $n = 8192$ and rate $R = 0.5$ (RD limit 0.11) for the given number of iterations.

Algorithm iterations	Distortion D
201	0.119437
167	0.12009
137	0.120855
113	0.121748
107	0.122088
103	0.122309

VI. Conclusions

In this paper, we presented two decimation approaches for WE that can significantly lower 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. We showed a significant reduction in complexity ($> 90\%$) compared to the scheme in [7], with only moderate losses in distortion. There are several features of the algorithm that can be improved, such as designing good convolutional protographs that permit shorter window sizes and optimized message passing schedules within a window. This, along with a comparison to other encoding algorithms, such as the BPGD algorithm, is the subject of ongoing work.

Acknowledgements

This work was supported in part by NSF grants CCF-1439465, CCF-1440001, and CCF11-61754.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] A. Golmohammadi, D. G. M. Mitchell, J. Kliewer, and D. J. Costello, "Windowed Encoding of Spatially Coupled LDGM Codes for Lossy Source Compression," *in Proc. IEEE Int. Symp. Inf. Theory*, Barcelona, Spain, July 2016. (http://web.nmsu.edu/~dgmm/SC-LDGM.pdf)

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