Improved Iterative Bit Flipping Decoding Algorithms for LDPC Convolutional Codes

Xin Sheng Zhou, Bruce F. Cockburn, Stephen Bates
Department of Electrical and Computer Engineering
University of Alberta
Edmonton, AB T6G 2V4, Canada
xzhou, cockburn, sbates@ece.ualberta.ca

Abstract—A new class of iterative bit flipping (BF) decoding algorithms adapted for low-density parity check (LDPC) convolutional codes is proposed. Compared with Gallager’s original BF algorithm, the new BF algorithms improve both the coding gain and the error correction speed. At a bit error rate (BER) of 10^{-4}, the best of the new bit-flipping algorithms achieves, after only 6 iterations and with much simpler decoding hardware, a coding gain within 3.5 dB of that of the conventional min-sum belief propagation decoding algorithm.

I. INTRODUCTION

Low-density parity check block codes (LDPC-BCs) were first proposed by Gallager in the early 1960’s [1]. However, the construction that he described was awkward, and the performance that he reported was not very impressive, hence this work was largely ignored for over thirty years. In 1996, LDPC-BCs were rediscovered by Mackay and Neal and shown to have error correction performance approaching the Shannon capacity limit [2]. In 1999, Jiménez Felström and Zigangirov proposed low density parity check convolutional codes (LDPC-CCs) [3]. They showed that LDPC-CCs have better code performance than regular LDPC-BCs under the same memory capacity constraints in the decoder. Furthermore, LDPC-CCs have a simpler encoder algorithm, and a hardware decoder can be implemented conveniently in silicon by a cascade of physically identical processors [3]. Moreover, LDPC-CCs allow data sequences of arbitrary length to be encoded, making these codes especially attractive in situations where either (a) the data block does not fit into the fixed payload field of the available data frame, or (b) the data is produced continuously by a streaming application [4].

To decode LDPC codes, the most powerful decoding algorithms process probability or “belief” information associated with the received bits. The magnitude of the received bit signals is represented by either floating-point or fixed-point numbers of some finite precision. In the so-called sum-product algorithm [5], an iterative real number calculation is performed. The resulting code performance can approach the Shannon Limit [2]. However, the computational complexity is rather large and hence the algorithm must be implemented in a custom application-specific integrated circuit (ASIC) to achieve the decoding throughput required in commercial applications [6]. Alternative algorithms have been proposed in order to decrease the computational complexity, such as the min-sum algorithm [7] and Kou’s bit-flipping (BF) algorithm [8], at the cost of weaker code performance.

Another algorithm for decoding LDPC codes is Gallager’s BF algorithm [1]. This algorithm uses only “hard-sliced” binary signals at the decoder inputs, and reliability information related to the actual analog magnitudes of the received bit signals is not considered. A bit is flipped if the number of unsatisfied parity check equations, for which the bit is an input, exceeds a fixed threshold value b. Although the BF algorithm is practical only for transmission well below the channel capacity or in high signal-to-noise (SNR) ratio applications, the decoding algorithm is simple and fast and the hardware implementation requires less circuitry and less power. Considering recent transmission technology improvements, BF algorithms could be competitive in modern reliable high-speed networks. However, relatively few improvements have been proposed for Gallager’s BF algorithm due to its already simple structure. In [9], Miladinovic and Fossorier proposed to randomly flip the suspect bits with pre-defined probability p based on the observation that flipping all of the unsatisfied bits, while not considering their reliability, would flip too many correct bits and hence degrade the decoding performance. They proposed to flip bits (data or parity) with probability p < 1 if those bits are inputs to unsatisfied parity constraints. That is, instead of flipping all of the unsatisfied bits, only a fraction of them would be flipped. In this way, fewer correct bits would be erroneously flipped and convergence on properly corrected data would be improved. The decoding performance would improve at the cost of a slower, more conservative decoding process in high SNR situations.

In this paper, we present simulation evidence that demonstrates that for a well-studied (128,3,6) LDPC-CC [4], choosing a strictly alternating threshold pattern of the form 3-2-3-2, that is a conservative b=3 for odd iterations and a more aggressive b=2 for even iterations, can not only improve the coding gain, it can also speed up decoder convergence. At a BER at 10^{-4}, what we call the 3-2 bit flipping pattern achieves a coding gain within 3.5 dB of that of the min-sum belief propagation decoding algorithm, with only 6 decoding iterations and with much simpler signals (and hence much simpler hardware and lower power in a silicon implementation). What is more, the coding gain under the same conditions is 2 dB greater than with Gallager’s BF algorithm.
II. GALLAGER’S ORIGINAL BIT FLIPPING ALGORITHM

In Gallager’s bit flipping algorithm, the decoder computes each parity check, using only hard-sliced binary input signals with simple XOR operations. It then schedules a bit to be flipped if the number of failed parity checks exceeds a fixed flipping threshold value \( b \). The flipped bits are then used in the next iteration of the decoding process. The decoding algorithm stops when either all of the parity checks are satisfied or a pre-defined maximum iteration limit is reached. The resulting simple BF algorithm is as follows:

**Step 1:** Compute the parity-check equations. If all of these parity constraints are satisfied, then stop decoding.

**Step 2:** Find the number of unsatisfied parity-check equations for each bit \( i \), denoted henceforth by \( f_i \).

**Step 3:** Consider each of the bits in turn. If all of the parity check equations with a particular bit as input are unsatisfied, then flip that bit prior to the next decoding iteration.

**Step 4:** Repeat steps 1 to 3 until all of the parity check equations are satisfied or until a predefined maximum iteration number is reached.

Compared with other LDPC decoding algorithms, BF algorithms use only one bit information and the parity check decision is based on a simple XOR operation. Because of its simplicity, a BF decoder could save large amount of power and silicon area. In addition, the decoding speed and throughput could be faster. Although BF algorithms only have good performance in high SNR, it could be a good candidate for power hungry mobile applications or gigabit high-speed networks.

III. BIT FLIPPING THRESHOLD PATTERNS

In an \((N,J,K)\) LDPC-CC, the previous \( N \) information bits and \( N \) parity check bits are stored in a First-In First-Out (FIFO) memory queue \([3]\). The encoder calculates the parity check equations by a simple XOR operation. Each bit is involved in \( J \) parity check equations and each parity check equation involves \( K \) bits. The bit positions for each parity check equation are changed, in a cyclic pattern, for each new parity check bit.

The bits are transmitted through a noisy communication medium and/or stored in an imperfect memory medium, and are hence corrupted by noise. Such noise is often modelled as Additive White Gaussian Noise (AWGN). At the receiving end, a hard decoder can be used to estimate the received bits as “0” or “1”. These binary estimates can then be sent to the BF algorithm for further processing that corrects errors.

With respect to a bit flipping decoding algorithm, we define the bit flipping threshold pattern \( b_1-b_2-b_3 \) to be the sequence of bit flipping thresholds that are used in the 1st, 2nd, 3rd, etc. iterations. If the pattern is shorter than the desired number of decoding iterations, then the sequence is repeated as often as necessary, starting again each time at \( b_1 \).

Gallager’s BF algorithm was developed to decode block-based LDPC codes, but here we need to decode convolutional LDPC codes. Essentially Gallager’s BF algorithm performs iterations in time where LDPC-CC decoders perform iterations in space over a cascade of pipelined decoder processors \([3]\).

Our new BF algorithms, modified for LDPC-CCs and based on different bit flipping threshold patterns, share the following structure:

**Step 1:** Within the buffer of data and parity bits inside each decoder processor, find the number \( f_i \) of unsatisfied parity-check equations for each bit \( i \) when that bit goes through the processor.

**Step 2:** At the end of each processor, if the unsatisfied parity check equation number of the output bit exceeds the bit flipping threshold \( b \), as specified by the bit flipping threshold pattern, then flip that bit before it goes into the next decoding processor.

**Step 3:** After the bit goes through all the decoding processors, the decoding process of that bit is finished and the decoded bit is sent out.

Because such a BF algorithm uses binary signals produced by a hard slicer, with signal strengths, probabilities or belief information being ignored, the decoding algorithm only performs well in a relatively high signal-to-noise ratio (SNR) environment. Fig. 1 shows BER simulations that suggest that BF decoding is effective only for \( Eb/No \) SNRs of above 4 dB. Each point on the plot corresponds to at least 100 error events. If the SNR is too low, then the decoder flips too many good bits in each iteration and the calculation does not converge properly or converges only very slowly.

The intuition behind the BF algorithms is that the greater the number of unsatisfied parity check equations involving a particular bit, the higher will be the likelihood that that bit is in error. The best bit flipping threshold value \( b \) is based on the parity check matrix characteristics, the SNR and the error bit density. If \( b \) is chosen to be too small (i.e., the algorithm is too aggressive), then more correct bits would be flipped and the algorithm will not converge on the correct data. On the other hand, the flipping threshold value \( b \) should be set to a sufficiently small value so that the error bits indicated only by fewer unsatisfied parity check equations can get flipped. If \( b \) is chosen to be too high (i.e., the algorithm is too conservative), then convergence on the correct data will be too slow or may get stuck in local minima with some errors left uncorrected.

For the benchmark \((128,3,6)\) LDPC-CC, each bit involves 3 parity check equations. The flipping threshold value \( b \) could be chosen to be between 3 and 2. For the first several iterations, \( b \) could be set to 3. After the error bit density falls below some threshold, \( b \) could be set to 2 to flip the bits involved in at least two unsatisfied parity check equations. The overall BER might then rise due to the smaller \( b \). Then \( b \) could be increased to 3 again to correct the bits and drop the error bit density to a lower level. The threshold value \( b \) can be shifted back and forth between 3 and 2 until some maximum iteration number is reached. In this paper, we assumed that the bit flipping threshold pattern is set to a fixed number of threshold value 3’s followed by a threshold value of 2, such as 3-2, 3-3-2, 3-3-3-2. This restriction was supported by experimental evidence from many simulation trials.
A. Decoding Performance

BF algorithms with different bit flipping threshold patterns for the same (128,3,6) LDPC-CC were simulated. To determine the maximum decoding performance capability of each algorithm, sixty decoding iterations were performed. As shown in Fig. 1, the repeating 3-2 bit flipping pattern achieves 2 dB greater coding gain, at a BER of $10^{-4}$, compared to Gallager’s all-3 bit flipping threshold pattern algorithm. Interestingly, the 3-2 pattern achieves a coding gain that is only 3.5 dB worse than the min-sum decoding algorithm, which uses much more computationally intensive belief propagation. Other bit flipping patterns (e.g. 3-3-2, 3-3-3-2 and 3-3-3-3-2) were also simulated for 60 iterations. Their curves show the same coding gain, so it appears that repeating the 3 threshold for more than one iteration produces little or no benefit.

B. Iteration Number

Fig. 2 shows the results of increasing the number of decoding iterations, assuming the same 3-2 pattern. There is indeed a gain in coding performance as the iteration number is increased up to 6, but the gain is very small beyond 6. For the (128,3,6) LDPC-CC, the curve for 6 iterations approaches within 0.2 dB of the curve for 60 iterations. It would appear that 6 iterations might be a good balance between decoding performance and computational complexity.

C. Error Correction Speed

In Fig. 3, algorithms with different bit flipping threshold patterns were simulated with 6 iterations. It is evident that for all of the considered SNR values, the 3-2 bit flipping threshold pattern has the lowest BER and the 3-3-3-3-2 pattern has the highest BER. In Fig. 4, the BER after different iterations for different bit flipping threshold patterns was simulated at the same SNR value of $Eb/No = 6$ dB. It appears that the BF threshold pattern 3-2 has the fastest error correction speed compared to its peers. Note that Gallager’s algorithm, with a fixed 3 threshold pattern, gets stuck at a relatively high BER. It appears that by alternating between the 3 and 2 thresholds, the decoding algorithm is able to escape from local minima. Repeating the 3 threshold does not produce faster convergence: the 3-2 pattern was found to produce the fastest convergence.

From the above simulation results, it can be concluded that for the (128,3,6) LDPC-CC BF decoding algorithm, using bit flipping threshold pattern 3-2 with 6 iterations achieves the
Fig. 4. BER of BF pattern 3-3-2-3-3-2-3-3-3-2 after different iterations, Eb/No=6dB

V. CONCLUSIONS

In this paper we presented simulation evidence that supports the effectiveness of a class of new iterative bit-flipping decoding algorithms for convolutional LDPC codes. Through experimentation we determined that the best performance was achieved when we strictly alternated between conservative iterations (where the threshold for flipping a bit is 3 failed parity checks) and more aggressive iterations (where the threshold is lowered to 2 failed parity checks). Using this particular threshold pattern (i.e., 3-2-3-2-) in Gallager’s BF algorithm, modified for the (128,3,6) LDPC-CC, not only improves the coding gain, it also speeds up the error correction convergence, and hence reduces the iteration number and the decoding computational complexity. Remarkably, the best bit-flipping decoding algorithm produced coding gain, at a BER of 10^{-4}, within 3.5 dB of the min-sum algorithm, which uses belief propagation and hence requires much more computation (and more complex decoder hardware).

Additional work is required to explore other variations on the bit-flipping technique applied to the decoding of LDPC-CCs. Also, more work is required to define which operating environments would best benefit from bit-flipping decoding. Power-constrained systems operating over a relatively clean channel would appear to be one good candidate for the application of the BF algorithms. Hybrid decoder designs might also be worth investigating, that is, decoders that fall back from full min-sum decoding to bit-flipping decoding when channel conditions allow it.

REFERENCES