LOW-COMPLEXITY FINITE PRECISION DECODERS FOR LOW-DENSITY PARITY-CHECK CODES

Graduate Student: **Shiva Kumar Planjery**
Advisors: **Bane Vasic, David Declercq, and Michael W. Marcellin**
Department of Electrical and Computer Engineering, University of Arizona, Tucson
ETIS ENSEA/UCP/CNRS, Cergy-Pontoise, France

Abstract

We present a new class of finite-precision decoders for low-density parity-check (LDPC) codes. These decoders are much lower in complexity compared to conventional floating-point decoders such as the belief propagation (BP) decoder, but they have the potential to outperform BP. The messages utilized by the decoders assume values (or levels) from a finite discrete set. We discuss the implementation aspects as well as describe the underlying philosophy in designing these decoders. We also provide results to show that in some cases, only 3 bits are required in the proposed decoders to outperform floating-point BP.

I. INTRODUCTION

Error-correcting codes are indispensable for any modern digital communication system which requires reliable transmission and/or storage of digital data. Over the past decade, a particular class of error-correcting codes called low-density parity-check (LDPC) codes that were originally discovered by Gallager in the 1960's, and rediscovered some thirty years later, has sparked a widespread interest and has been a subject of intense research in the field of communications. The traditional algorithms used for decoding LDPC codes are primarily based on an iterative algorithm called belief propagation (BP), which operates on the graphical model of the code. Under BP decoding, these codes were shown to asymptotically perform close to the theoretical limit established by Shannon's noisy channel coding theorem [2]. The remarkable performance of LDPC codes as well as their simple and efficient high-speed implementations have made them very attractive for use in a plethora of applications ranging from wireless communication and deep-space communcation systems to magnetic storage media.

In the past several years, a considerable amount of research has been dedicated towards constructing capacity-achieving LDPC codes that have good distance properties, and finding better iterative decoders that enable simpler hardware implementations. Richardson et. al. in [2] proposed the technique of density evolution under BP decoding in order to determine the asymptotic decoding threshold of a particular code and suggested using this analysis in order to optimize the code's profile for the best possible decoding threshold. Although the density evolution approach can provide a fairly accurate prediction of the performance of a code in the very low signal-to-noise ratio (SNR) region (or waterfall region), it cannot predict in the higher SNR region (or error floor region). This is because the density evolution approach does not take into account the finite-length effects of the code as it assumes an infinite code length and a node in the graph representation of the code considers the graph to be a tree during decoding. Therefore, BP decoding is suboptimal for practical implementations of LDPC codes. In addition, the problem of error floors exists in LDPC codes of practical lengths.

E-mail: {shivap, vasic, marcellin}@ece.arizona.edu, declercq@ensea.fr

Error floor is an undesirable phenomenon typically present in iterative decoding based codes, where an abrupt degradation in the error-rate performance of the code occurs in the high SNR region. The causes of error floor can be attributed to the presence of certain harmful structures in the Tanner graph of the code called *trapping sets* that cause the decoder to fail for error patterns of low weight. The notion of trapping sets was first introduced by Richardson [4] in order to characterize the failures of iterative decoders. These trapping sets can be present in any finite-length code even though it has been optimized for a good decoding threshold. Hence, LDPC codes that are optimized using the density evolution approach can still exhibit high error floors.

Recently, the design of quantized iterative decoders having low complexity implementations, has gained prominence due to increasing speed requirements and stricter hardware constraints for practical realizations. In this regard, the problem of designing quantized BP and min-sum decoders has been investigated [2], [5], [6]. These proposed quantization schemes are primarily based on achieving the best possible asymptotic decoding threshold using density evolution, and approaching the performance of floating-point BP, i.e., minimizing the loss in performance due to quantization. Again for reasons mentioned previously, these schemes do not guarantee a good performance on a finite-length code especially in the low-noise regime. In addition, effects of quantization can contribute further to the error floor phenomenon (which is neglected in the designs).

In this paper, we present a new class of decoders for the binary symmetric channel (BSC) that addresses both finite precision as well as the error floor phenomenon. These decoders are obtained with the purpose of improving the message-passing process on finite-length graphs as BP decoding is suboptimal on finite-length graphs. This is carried out by using certain subgraphs as combinatorial objects that could potentially be trapping sets and deriving finite precision decoders that reduce the failure rate on these subgraphs. Hence, these decoders have the potential to outperform the floating-point BP decoder in spite of using finite precision to represent messages. At the same time these decoders greatly simplify the hardware implementation without compromise in performance.

The rest of the paper is organized as follows. Section II provides the necessary preliminaries for this work. In Section III, we provide a description of the low-complexity finite-precision decoders and briefly discuss their implementation aspects as well as the design methodology which is based on analyzing decoding on isolated subgraphs. We finally provide numerical results and conclusions in Section IV.

II. PRELIMINARIES

In this section, we shall provide the necessary fundamentals related to LDPC codes and briefly describe the BP algorithm. We shall also elaborate on the notion of trapping sets which are essential for deriving good decoders and introduce some notations.

A. LDPC codes

LDPC codes are linear codes that are characterized by a sparse parity check matrix H containing a small number of nonzero entries. These codes can be conveniently represented by bipartite graphs called Tanner graphs, which are more useful representations when carrying out the iterative decoding process. The Tanner graph representation of an LDPC code consists of two sets of nodes. One set of nodes are called variable nodes and they represent the bits associated with a codeword. The other set of nodes are called check nodes and they represent parity check constraints on the bits of the codeword

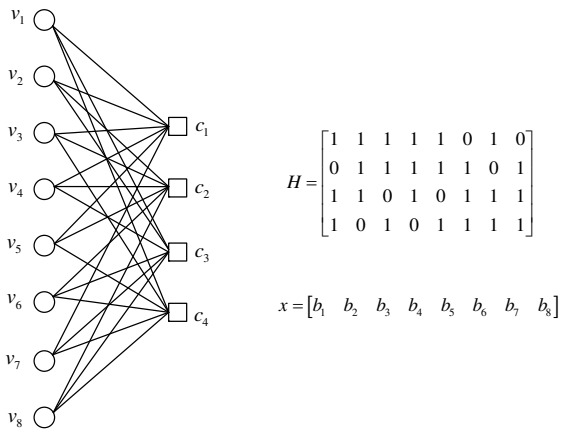


Fig. 1. Example of a rate 1/2 (3,6) LDPC code

which are defined by the parity check matrix H. An edge of the graph connects a variable node to a check node only if that particular bit represented by the variable node participates in the parity check equation represented by the check node. The degree of a node is the number of neighbors it is connected to in the graph. Regular LDPC codes are codes for which all the variable nodes have the same degree and all check nodes have the same degree, whereas irregular LDPC codes can have different degrees for different nodes.

Figure 1 shows the example of a Tanner graph for a rate-half (3, 6) binary LDPC code with code length 8, and its corresponding parity check matrix. The circles on the Tanner graph denote variable nodes and the boxes denote check nodes. The vector x consists of bits b_1 to b_8 that are associated to their corresponding variable nodes v_1 to v_8 on the Tanner graph. x is a codeword if and only if it satisfies the matrix product, $Hx^T = 0$ (satisfiability condition). This implies that every row of the parity check matrix H corresponds to a parity check constraint on the bits of a codeword. Then all the bits represented by the variable nodes that are connected to a particular check node must add up to zero modulo 2, and under such condition, the check node is considered to be satisfied. If the parity check equation does not hold, then the check node is considered to be unsatisfied.

We shall adopt notations used in [7]. Let $G = (V \cup C, E)$ denote the Tanner graph of a binary LDPC code $\mathcal{C}$ with the set of variable nodes $V = \{v_1, \cdots, v_n\}$ and set of check nodes $C = \{c_1, \cdots, c_m\}$. E is the set of edges in G. The code has length n and code rate R. For a vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, the support of $\mathbf{v}$ denoted as $\mathrm{supp}(\mathbf{v})$, is defined as the set of all variable nodes such that $v_i \neq 0$. A code $\mathcal{C}$ is said to be d_v-left-regular if all variable nodes in V of graph G have the same degree d_v. Let $\mathbf{r} = (r_1, r_2 \ldots, r_n)$ be the input to the decoder from the BSC.

B. Decoding algorithms

Several message-passing algorithms exist in literature that are used for decoding LDPC codes. We briefly describe the BP algorithm in order to highlight the distinction between the currently proposed decoders and existing message-passing decoders, since most decoders are based on the BP algorithm or its modified version.

Any message-passing algorithm for LDPC codes can be described by defining update rules at the variable node and check node. The variable node update rule and the check node update rule, denoted as functions Ψ_v and Ψ_c resepctively, are used to determine the outgoing messages. Initially, every variable node v_i calculates its channel value y_i based on the observation value r_i received from the channel. For the BP algorithm implemented in the "log-liklihood domain", the channel value y_i is a log-liklihood ratio calculated from the observation r_i. Assuming that a transmitted bit b_i can be a zero or a one with equal probability, the value y_i can be calculated as

$$\log \frac{p(r_i|b_i = 0)}{p(r_i|b_i = 1)}$$

The calculation of y_i depends on the type of channel and noise distribution.

Let $m_1, \cdots, m_{d_v-1}$ denote incoming messages to a variable node of degree d_v and $m_1, \cdots, m_{d_c-1}$ denote the incoming messages for a check node of degree d_c. Note that while determining the outgoing message on any particular edge of a node, the incoming message on that particular edge is not included in the computation of the outgoing message. This is to ensure that the outgoing message is an *extrinsic message* and the dependencies between the messages entering the node are reduced.
For the BP algorithm, the variable node update and check node update rules are defined as

$$\begin{aligned} \Psi_v(y_i, m_1, \ldots, m_{d_v-1}) &= \sum_{j=1}^{d_v-1} m_j + y_i \\ \Psi_c(m_1, m_2, \ldots, m_{d_c-1}) &= 2\tanh^{-1}\left(\prod_{j=1}^{d_c-1} \tanh\left(\frac{m_j}{2}\right)\right) \end{aligned}$$

Initially all messages are set to zero and each variable node sends its channel value as the outgoing message. Messages are then passed iteratively between check nodes and variable nodes using the update rules Ψ_c and Ψ_v.

At the end of each iteration, a hard decision rule is carried out at the variable node which determines whether its associated bit is a one or a zero based on the values of the incoming messages and the channel value y_i. For the BP algorithm, the bit b_i is decided by taking the sum of all the incoming messages and the channel value, and observing the sign of the result. These bits obtained from the hard decision rule are then sent to the check nodes along the edges of the graph in order to verify if the decoder has converged to a codeword. If so, the iterative process is terminated, else the iterative process is continued until a maximum number of iterations is reached.

C. Trapping sets

Trapping sets are structures present in the Tanner graph of the code that cause the decoder to fail for error patterns of low weight, usually much lower than the error-correction capability of the code under maximum likelihood (ML) decoding. Following the definition given by Richardson in [4], a trapping set $\mathbf{T}(\mathbf{r})$ is a non-empty set of variable nodes in G that are not eventually corrected by the decoder for a particular input $\mathbf{r}$. A standard notation used to denote a trapping set is (a, b) where $a = |\mathbf{T}(\mathbf{r})|$, and b is the number of odd-degree checks present in the induced subgraph of $\mathbf{T}(\mathbf{r})$. The critical number of a trapping set is the minimum number of variable nodes that have to be initially in error for the decoder to end up in the trapping set. The critical number conveys how harmful a given trapping set is. The lower the critical number, the more harmful the trapping set.

III. LOW-COMPLEXITY FINITE PRECISION DECODERS

In this section, we present a new class of finite precision decoders that are much lower in complexity compared to the BP decoder. For the proposed decoders, the update rules do not mimic the rules used in the BP algorithm; they are instead derived using knowledge of trapping sets that are already known for traditional decoders such as Gallager-B or BP. The rules can be described algbraically or using Boolean functions (or look-up tables). For sake of exposition, we shall begin by providing an algebraic description of the decoders [7].

A. Description of decoders

For these decoders, the messages take values from a finite discrete set $\mathcal{M} = \{-L_k, \cdots, -L_2, -L_1, 0, L_1, L_2, \cdots, L_k\}$, where $L_i \in \mathbb{R}^+$ and k is the number of bits used for representation. The sign of a message $\mu \in \mathcal{M}$ represents the message's estimate of whether the associated bit is zero or one, and the magnitude $|\mu|$ represents the reliability measure of its estimate. Also $\mathcal{M}$ is defined such that $L_i > L_j$, for any $i > j$. The set $\mathcal{Y} = \{\pm \mathrm{C}\}$ denotes the set of possible channel values. For each variable node v_i in G with r_i received from the BSC, the channel value $y_i \in \mathcal{Y}$ is computed as $y_i = (-1)^{r_i}\mathrm{C}$. The value C gives a measure of how much the decoder relies on the channel's estimate of the bit node.
At the check node, the update function Ψ_c is defined as

$$\Psi_c(m_1, \ldots, m_{d_c-1}) = \left(\prod_{j=1}^{d_c-1} \mathrm{sgn}(m_j) \right) \min_{j \in \{1,\ldots,d_c-1\}} (|m_j|)$$

where sgn denotes the standard signum function.
At the variable node, the update function Ψ_v is defined as

$$\Psi_v(y_i, m_1, \ldots, m_{d_v-1}) = Q\left(\sum_{j=1}^{d_v-1} m_j + \Omega(m_1, \ldots, m_{d_v-1}) \cdot y_i \right)$$

where Ω is symmetric function $\Omega : \mathcal{M}^{d_v-1} \rightarrow \{0, 1\}$ that could be linear or nonlinear. The function Q is a quantization function that compares the sum of the incoming messages and y_i with a set of thresholds defined by a threshold set $\mathcal{T} = \{T_1, T_2, \cdots, T_M\}$, where $T_i \in \mathbb{R}^+$, and for any $T_p, T_q \in \mathcal{T}$, $T_p > T_q$ if $p > q$. The function $Q(x)$ outputs the message $\pm L_i$ if $T_i \leq |x| < T_{i+1}$.

Note that if the function Ω is nonlinear, these decoders are different from any existing quantized message-passing decoders. The function Ψ_v can now be uniquely described by specifying the channel output set $\mathcal{Y}$, message set $\mathcal{M}$, threshold set $\mathcal{T}$, and the function Ω, which constitute the design parameters for these decoders. The non-linearity introduced into the function allows the variable node to capture some of its local neighborhood that could be potentially harmful (ex: if the node is in a six cycle), and accordingly compute outgoing messages that help the decoder to converge. More details on this shall be provided in the next subsection while discussing the design methodology.

B. Implementation aspects

Although we have represented messages as levels from the set $\mathcal{M}$, they are represented as binary vectors of length k during implementation. There is a one-to-one correspondence between the binary vector representation and the levels L_i defined in set $\mathcal{M}$. For example, consider a decoder that uses a message set $\mathcal{M}$ with 7 levels. Each level L_i in the set $\mathcal{M}$ is represented by a 3-bit binary vector. The most significant bit (MSB) denotes the estimate of whether the associated node is one or zero, i.e., the sign

of L_i. All positve levels have an MSB of zero and all negative levels have an MSB of 1. The next two bits denote how reliable the estimate is. For example, L_3 is the strongest possible level with an estimate of zero and hence its 3-bit representation is 011. In this manner, the mapping from the levels L_i in set $\mathcal{M}$ to their 3-bit binary representation can be derived and is shown in table I.

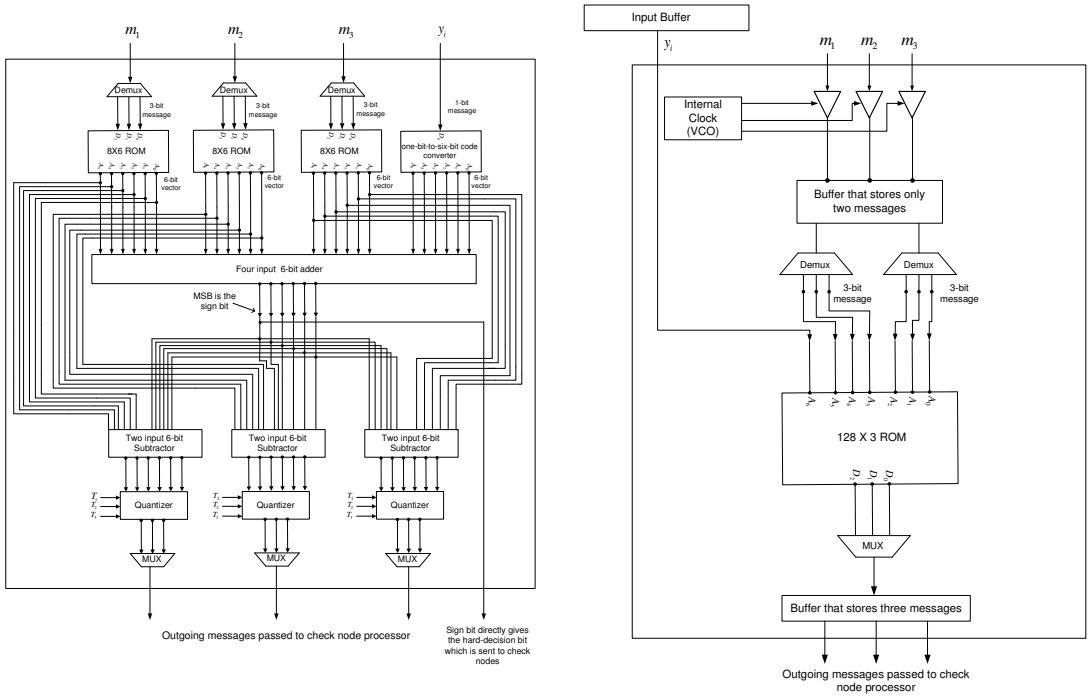
Levels	3-bit representation
L_3	011
L_2	010
L_1	001
0	000 or 100
$-L_1$	101
$-L_2$	110
$-L_3$	111

TABLE I
3- BIT BINARY VECTOR REPRESENTATION OF LEVELS

Similarly, there exists a one-to-one correspondence between the algebraically defined Ψ_v and a simple look-up table. Depending on the application and type of requirements in the decoder realization, either form can be used for implementation. In the algebraic form, Φ_v is implemented based on the message set $\mathcal{M}$, threshold set $\mathcal{T}$, channel output set $\mathcal{Y}$ and channel weight function Ω. For simplicity of exposition, we shall assume $\Omega = 1$ and consider only decoders for 3-left-regular codes. In order to facilitate the implementation in algebraic form, the messages which are represented as 3-bit binary vectors must be converted to binary vectors in 2's complement form which represent the actual values of L_i. Since we are using the 2's complement form, we will need extra bits to represent the sign, integer and fractional parts of the values of L_i. Care must be taken in choosing the values for L_i so that minimal number of extra bits are required while converting to the 2's complement form. Figure 2(a) shows the general schematic for implementation using the algebraic form of Ψ_v. Figure 2(b) shows the implementation using the look-up table form. The look-up table corresponding to Ψ_v stored in a ROM is used to determine the outgoing message. Due to larger memory requirements in this implementation, outgoing messages are computed sequentially in this scheme instead of parallel so that only a single ROM is required for each variable node. Again the messages can be computed parallely using multiple ROMs of the look-up table if hardware area is not a constraint. The tri-state buffers (represented as triangles) in Figure 2(b) ensure that only extrinsic messages are calculated.

Based on the implementation schemes described we can point out two advantages that make implementation using the algebraic form an attractive choice over look-up table form. Firstly, the implementation scheme using algebraic form is simple and straightforward and if the modules of the implementation use reconfigurable components, then changing the update function Ψ_v can be easily done by simply changing the thresholds and magnitudes. This is advantageous especially for the case of a decoder that switches its variable node update function to another and also for enabling flexibility and reconfigurability to modify decoders in the hardware whenever the need arises. Secondly, for decoders that use larger number of bits (say 4 or 5 bits), the look-up table of Ψ_v can become quite large leading to large memory requirements and utilization of hardware area. On the other hand, the complexity will only linearly increase with the number of bits for the implementation using algebraic form.

However, if we are to strictly use only n bits for the n-bit decoders to represent the messages, the look-up table form needs to be used. The look-up table form may especially be well suited when the



(a) (b)

Fig. 2. Variable node processor: (a) Algebraic form; (b) Look-up table form

Ω function is nonlinear and the message set is small, since there is no added complexity involved for implementing Ω in the look-up table form. Also the look-up table form may be helpful for faster search of good update rules. Either of the two implementations can be used based on the hardware constraints, decoder speed requirements, and type of application that it is used for. As an example, Table II shows the look-up table form corresponding to a Ψ_v for a 7-level decoder which is defined by a message set $\mathcal{M}$ ={−8.5,−3.5,−1,0,1,3.5, 8.5}, threshold set $\mathcal{T} = \{1, 3.5, 8.5\}$, C = 1.5, and function $\Omega = 1$. m_1 and m_2 are incoming messages to a degree-3 variable node and m_o is the outgoing message. Note that with deeper introspection into the look-up table, there may exist simple Boolean functions that have even lower complexity than algebraic form.

C. Design methodology

A key strategy used in deriving good update rules is to analyze decoding on isolated subgraphs that could be potential trapping sets for a given decoder. Since we consider symmetric decoders, we can assume that the all-zero codeword is transmitted during analysis. In order to decode on an isolated subgraph, we assume that all the variable nodes outside the subgraph are initially correct (receive initially correct channel values) and that the neighborhood of the subgraph is such that the messages entering into the subgraph from outside are not affected by the messages being propagated within the subgraph. As an example, Figure 3 illustrates the decoding on an isolated eight-cycle which is a potential (4,4) trapping set. The nodes with solid lines are the nodes that belong to the subgraph. The open boxes represent degree-2 check nodes and the filled boxes represent the degree-one check nodes in the subgraph. Clearly the subgraph has 4 variable nodes and 4 degree-one check nodes, hence a (4,4) trapping set. The nodes with dotted lines represent the variable nodes outside the subgraph.

m_1	m_2	y_i	m_o
L_1	L_1	C	L_2
L_1	L_1	$-C$	0
L_1	L_2	C	L_2
L_1	L_2	$-C$	L_1
L_1	L_3	C	L_3
L_1	L_3	$-C$	L_2
L_1	0	C	L_1
L_1	0	$-C$	0
L_1	$-L_1$	C	L_1
L_1	$-L_1$	$-C$	$-L_1$
L_1	$-L_2$	C	$-L_1$
L_1	$-L_2$	$-C$	$-L_2$
L_1	$-L_3$	C	$-L_2$
L_1	$-L_3$	$-C$	$-L_3$

m_1	m_2	y_i	m_o
L_2	L_2	C	L_3
L_2	L_2	$-C$	L_2
L_2	L_3	C	L_3
L_2	L_3	$-C$	L_3
L_2	0	C	L_2
L_2	0	$-C$	L_1
L_2	$-L_1$	C	L_2
L_2	$-L_1$	$-C$	L_1
L_2	$-L_2$	C	L_1
L_2	$-L_2$	$-C$	$-L_1$
L_2	$-L_3$	C	$-L_2$
L_2	$-L_3$	$-C$	$-L_2$
L_3	L_3	C	L_3
L_3	L_3	$-C$	L_3

m_1	m_2	y_i	m_o
L_3	0	C	L_3
L_3	0	$-C$	L_2
L_3	$-L_1$	C	L_3
L_3	$-L_1$	$-C$	L_2
L_3	$-L_2$	C	L_2
L_3	$-L_2$	$-C$	L_2
L_3	$-L_3$	C	L_1
L_3	$-L_3$	$-C$	$-L_1$
0	0	C	L_1
0	0	$-C$	$-L_1$
0	$-L_1$	C	0
0	$-L_1$	$-C$	$-L_1$
0	$-L_2$	C	$-L_1$
0	$-L_2$	$-C$	$-L_2$

m_1	m_2	y_i	m_o
0	$-L_3$	C	$-L_2$
0	$-L_3$	$-C$	$-L_3$
$-L_1$	$-L_1$	C	0
$-L_1$	$-L_1$	$-C$	$-L_2$
$-L_1$	$-L_2$	C	$-L_2$
$-L_1$	$-L_2$	$-C$	$-L_2$
$-L_1$	$-L_3$	C	$-L_2$
$-L_1$	$-L_3$	$-C$	$-L_3$
$-L_2$	$-L_2$	C	$-L_2$
$-L_2$	$-L_2$	$-C$	$-L_3$
$-L_2$	$-L_3$	C	$-L_3$
$-L_2$	$-L_3$	$-C$	$-L_3$
$-L_3$	$-L_3$	C	$-L_3$
$-L_3$	$-L_3$	$-C$	$-L_3$

TABLE II
LOOK-TABLE FORM OF Ψ_v USED IN A 3-BIT DECODER

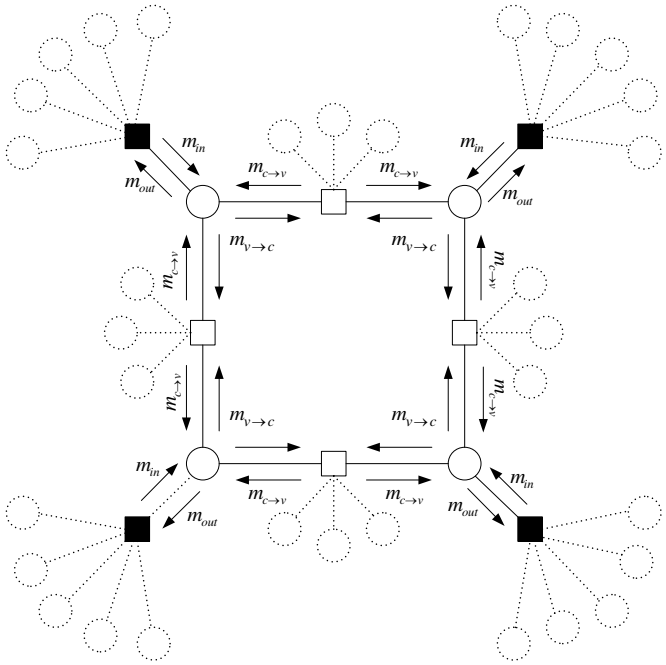


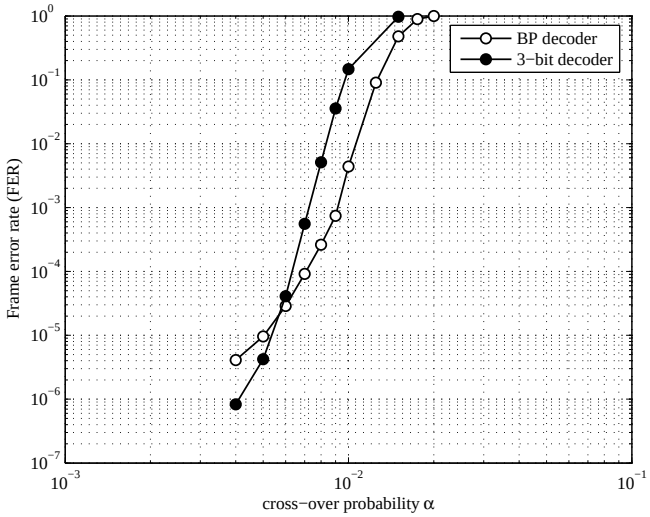
Fig. 3. Decoding on an isolated potential (4,4) trapping set

In Figure 3, $m_{v\to c}$ denotes messages going from variable node to the degree-2 check node and $m_{c\to v}$ denotes messages going from the degree-2 check nodes to the variable nodes. m_{in} denotes the outgoing message from the degree-one check node to a variable node. m_{out} denotes the outgoing message from a variable node to a degree-one check node. The messages $m_{v\to c}$ and m_{out} are computed using a specific variable node update table Ψ_v. However, in order to compute m_{in} at the end of every iteration, a different rule is needed assuming that all variable nodes outside the subgraph are initially correct. By the isolation assumption on the neighborhood of the degree-one check nodes, the degree-one check node in the subgraph will send the message $m_{in} = L_i$ into the subgraph during the i^{th} iteration until m_{in} reaches the maximum possible level L_k and thereafter it will always send the strongest message L_k. The remaining variable nodes and degree-2 check nodes follow the usual update rules and the decoding process is continued. Based on the error pattern in the subgraph, certain variable nodes in the subgraph will be initially wrong but may eventually become right by choosing a good variable node update table Ψ_v.

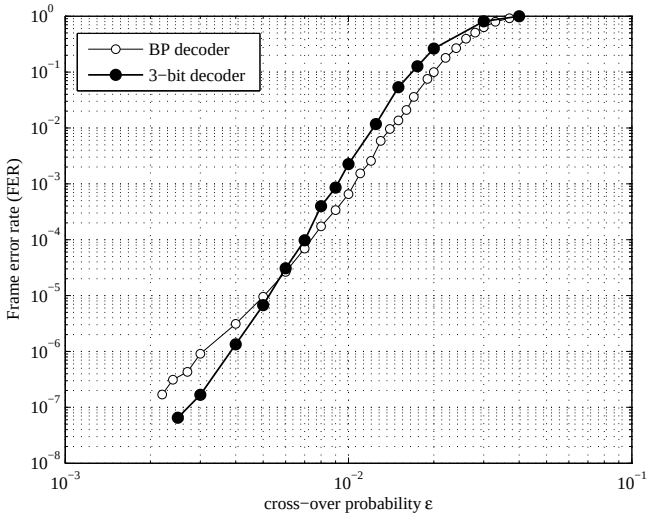
Using the technique of decoding on isolated subgraphs, the general method which is based on reducing failure rates on potential trapping sets can be summarized as follows. A database containing all possible subgraphs that are potential trapping sets is generated and called trapping set database. This database can be generated using analytical methods, by simulation or emulation of a decoder(s) on a given channel, or even by a combination of the simulation and analytical method. For example, the database could be generated by observing the failures in the high SNR region for a particular decoder or several decoders on a specific channel and using the subgraphs corresponding to these failures. Or, the database can be generated as previously mentioned by using a combinatorial construction algorithm and then further including some subgraphs associated with failures obtained during simulation of decoder(s). Essentially, the trapping set database contains subgraphs that have potential to be trapping sets for any given decoder, and then the decoders are designed such that they have reduced failure rates on these subgraphs. The goal is to search for a decoder that can correct most or all of the trapping sets (with error induced on them) under the isolation assumption. Some key parameters in the case of BSC that are used in the design are an increase in critical number of the potential trapping sets and convergence within few iterations when the decoder does converge. This process gives good decoders that are well-equipped to handle potentially harmful structures and helps improve the iterative decoding process on the graph. For a more rigorous explanation on the concept of isolation assumption, refer to [7].

IV. Numerical results and conclusions

We provide numerical results in order to validate our approach and illustrate that in some cases, only 3 bits are required for these decoders to surpass floating-point BP. Simulations of the BP decoder and 3-bit decoder specified by table II were carried out for frame error rate (FER) on two quasicyclic codes of different lengths: 1) $n = 5184$, $R = 0.834$, quasicyclic code, and 2) $n = 804$, $R = 0.75$, quasicyclic code. For both codes, the 3-bit decoder outperforms the floating-point BP in the error floor region with minimal loss in the waterfall region. Notice the difference in the slope of the FER curves in the error floor region for both decoders. Moreover, the 3-bit decoder achieves this at a fraction of the complexity of the BP decoder since both decoders used a maximum number of 100 iterations for decoding. Also the same 3-bit decoder that was derived using knowledge of trapping sets appears to be good on both codes. This suggests that the proposed decoders are not code-specific and the update rules derived appear to improve the message-passing process on a finite-length code by considering potentially harmful neighborhoods of nodes into the decoding.



(a)



(b)

Fig. 4. FER results: (a) $n = 5184$, $R = 0.834$, quasicyclic code; (b) $n = 804$, $R = 0.75$, quasicyclic code

REFERENCES

[1] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[2] T. Richardson and R. Urbanke, "Capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory* , vol 47, pp. 599–618, Feb. 2001.

[3] T. Richardson, A. Shokrollahi, R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619–637, Feb. 2001.

[4] T. Richardson, " Error floors of LDPC codes", in *Proc. of 41st Annual Allerton Conf. on commun., control and computing*, 2003.

[5] J. K. Lee and J. Thorpe, "Memory-efficient decoding of LDPC codes," in *Proc. Int. Symp. on Inform. Theory (ISIT 2005)*, Adelaide, Australia, pp. 459–463., Sept. 2005.

[6] B. Smith, F. R. Kschischang and W. Yu, "Low-density parity-check codes for discretized min-sum decoding," in *Proc. 23rd Biennial Symp. on Commun.*, pp. 14–17, 2006.

[7] S. K. Planjery, D. Declercq, S. K. Chilappagari, and B. Vasic, "Multilevel decoders surpassing belief propagation on the binary symmetric channel," in *Proc. Int. Symp. on Inform. Theory (ISIT 2010)*, Austin, TX, June 2010.