Finite Alphabet Iterative Decoders, Part I: Decoding Beyond Belief Propagation on the Binary Symmetric Channel

Shiva Kumar Planjery, Student Member, IEEE, David Declercq, Senior Member, IEEE, Ludovic Danjian, and Bane Vasic, Fellow, IEEE

Abstract—We introduce a new paradigm for finite precision iterative decoding on low-density parity-check codes over the binary symmetric channel. The messages take values from a finite alphabet, and unlike traditional quantized decoders which are quantized versions of the belief propagation (BP) decoder, the proposed finite alphabet iterative decoders (FAIDs) do not propagate quantized probabilities or log-likelihoods and the variable node update functions do not mimic the BP decoder. Rather, the update functions are maps designed using the knowledge of potentially harmful subgraphs that could be present in a given code, thereby rendering these decoders capable of outperforming the BP in the error floor region. On certain column-weight-three codes of practical interest, we show that there exist FAIDs that surpass the floating-point BP decoder in the error floor region while requiring only three bits of precision for the representation of the messages. Hence, FAIDs are able to achieve a superior performance at much lower complexity. We also provide a methodology for the selection of FAIDs that is not code-specific, but gives a set of candidate FAIDs containing potentially good decoders in the error floor region for any column-weight-three code. We validate the code generality of our methodology by providing particularly good three-bit precision FAIDs for a variety of codes with different rates and lengths.

I. INTRODUCTION

At the heart of modern coding theory lies the fact that low-density parity-check (LDPC) codes [1] can be efficiently decoded by message-passing algorithms which are based on the belief propagation (BP) algorithm [2]. The BP algorithm operates on a graphical model of a code known as the Tanner graph, and computes marginals of functions on the graph. While inference using BP is exact only on loop-free graphs (trees) and exact inference on loopy graphs is hard even under strong restrictions of graphical model topology, the BP still provides surprisingly close approximations to exact marginals on loopy graphs.

However, the sub-optimality of BP on loopy graphs contributes to the error floor phenomenon of LDPC codes. The error floor is an abrupt degradation in the slope of the error-rate performance in the high signal-to-noise-ratio (SNR) regime, where certain harmful loopy structures, generically termed as trapping sets [3] present in the Tanner graph of the code, cause the decoder to fail. Although there have been some important works related to devising algorithms that can provide better approximations on the marginals such as the generalized BP algorithms of [4], these are still too complex for practical use. Moreover, the effects of finite precision that are introduced when decoders are realized in hardware can further contribute to the error floor problem. A glimpse at the iterative decoders developed so far reveals a wide range of decoders of varying complexity. The simple binary message-passing algorithms such as the Gallager A/B algorithms [1] occupy one end of the spectrum, while the BP algorithm lies at the other end. The gamut of decoders filling the intermediate space can simply be understood as the implementation of the BP (and variants) at different levels of precision.

In this paper, which serves as the first part of our two-part paper series, we introduce a novel approach to the design of finite precision iterative decoders for the binary symmetric channel (BSC) which we refer to as finite alphabet iterative decoders (FAIDs) to signify the fact that the messages belong to a finite alphabet. Some of the key features that clearly distinguish our approach from all other existing works on finite precision iterative decoders (which shall be discussed in greater detail in the next section) are: 1) the messages are not quantized values of log-likelihoods or probabilities, and 2) the variable node update functions are simple well-defined maps rather than approximations of the update functions used in BP. The maps for variable node update in FAIDs are designed with the goal of increasing the guaranteed error-correction capability by using the knowledge of potentially harmful subgraphs that could be present in any given code, thereby improving the slope of the error floor on the BSC [5]. Since the variable nodes in the proposed decoders are now equipped to deal with potentially harmful neighborhoods, which is in contrast to BP which treats the loopy Tanner graph as a tree, the proposed decoders are capable of surpassing the floating-point BP in the error floor.

We restrict our focus to FAIDs for column-weight-three codes. The main reason for this is that such codes, while enabling extremely simple hardware implementations, are notoriously prone to higher error floors especially at moderate to high code rates compared to codes of higher column weights. Being able to design good yet simple decoders for these codes not only validates our novel approach, but also further confirms the importance of addressing the error problem from the viewpoint of improving the iterative decoding rather than one of code construction [6], [7], as it becomes very difficult.
to construct such codes to be devoid of certain harmful loopy graphs without compromise on the code rate and length.

We also provide a semi-heuristic-based selection method that is not code-specific, but gives a set of candidate FAIDs that contains potentially good decoders in the error floor, i.e., decoders that are likely to have an improved error floor performance on the code compared to floating-point BP. The method relies on analyzing the behavior of different FAIDs on a small number of carefully selected subgraphs which are potential trapping sets with errors introduced in the nodes of the subgraph. In order to carry out this analysis, we introduce the notion of noisy trapping set, which is a generalized notion that takes into account the possible effects of a neighborhood by initializing different possible sets of messages into the trapping set. Using this approach, we obtain a set of particularly good 3-bit precision (requiring only 3 bits to represent the ping set) FAIDs that are each capable of surpassing BP on several column-weight-three codes.

II. FUTURE PRECISION ITERATIVE DECODING AND THE ERROR FLOOR: PRIOR WORK

There have been several important works related to the design of quantized BP decoders. Early works include the Gallager-E and other finite precision decoders proposed by Richardson and Urbanke [8], and reduced-complexity BP decoders such as the normalized min-sum and offset min-sum decoders proposed by Chen et al. [9] and by Fossorier et al. [10]. More recent works include the quantized BP decoders proposed by Lee and Thorpe [11], and also by Kurkoks and Yagi [12]. In all of the aforementioned works, the quantization schemes are designed based on optimizing for the best decoding threshold on a given code using the asymptotic technique of density evolution (DE) [8] with the primary goal of approaching the performance of the floating-point BP algorithm. Since asymptotic methods are inapplicable to finite-length codes as they do not take their particular structures into account, these decoders designed for the best DE thresholds do not guarantee a good performance on a finite length code especially in the high SNR region. This was also evidenced in [13], [15] where the FAIDs that were chosen for a given code solely based on their DE thresholds were not the best performing FAIDs, and in some cases, even exhibited high error floors. Moreover, the effects of quantization on the error floor can vary depending on the particular structure of a given code [3], [16].

The error floor problem of LDPC codes has gained significant attention over the past several years with several works addressing the problem from a decoding perspective by proposing modifications to the BP decoding algorithm. Some of the notable works include augmented belief propagation [17], informed dynamic scheduling [18], multi-stage decoding [19], averaged decoding [20], and the use of post-processing to lower the error floors [21], [22]. While all these schemes certainly provide performance enhancements in the error floor, all of them require either a considerable increase in decoding complexity due to the modifications and post-processing or are restricted to a particular code whose structure is well known. In addition, they do not take finite precision into account. This can drastically affect the performance gains when implemented in hardware due to possible numerical precision issues [24]–[26].

Therefore, addressing the error floor problem while also taking finite precision into account in the decoder design is of prime importance especially with emerging applications in communication and data storage systems now requiring very low error rates and faster processing speeds. In this regard, there are some relevant works worth mentioning. Zhao et al. in [23] proposed several modifications to the offset min-sum decoder while taking quantization into account. Their proposed quantization schemes enabled the offset-min sum decoder to approach the performance of the floating-point BP algorithm on the additive white Gaussian noise channel (AWGNC) with six bits of quantization, with possible improvement in the error floor on certain codes. Zhang et al. in [24] also designed quantization schemes for BP over the AWGNC that led to substantial error floor reductions with six bits of quantization. More recently, Zhang and Siegel proposed a quasi-uniform quantization scheme in [27] and [28] for the min-sum and the BP algorithms for which the error floor performance could be improved with six bits of quantization.

However, using our novel approach, which is different from the above mentioned works (due to the reasons mentioned in Section I), we are able to design simple 3-bit precision decoders that are capable of surpassing the floating-point BP in terms of the error-rate performance.

III. PRELIMINARIES

Let $G$ denote the Tanner graph of an $(N, K)$ binary LDPC code $C$ of rate $R = K/N$, which consists of the set of variable nodes $V = \{v_1, \ldots, v_N\}$ and the set of check nodes $C = \{c_1, \ldots, c_M\}$. The degree of a node in $G$ is the number of its neighbors in $G$. A code $C$ is said to have a regular column-weight $d_c$ if all variable nodes in $V$ have the same degree $d_c$. The set of neighbors of variable node $v_i$ is denoted as $N(v_i)$, and the set of neighbors of check node $c_j$ is denoted by $N(c_j)$. The girth of $G$ is the length of shortest cycle present in $G$.

Let $x = (x_1, x_2, \ldots, x_N)$ denote a codeword of $C$ that is transmitted over the BSC, where $x_i$ denotes the value of the bit associated with variable node $v_i$, and let the vector received from the BSC be $r = (r_1, r_2, \ldots, r_N)$. Let $e = (e_1, e_2, \ldots, e_N)$ denote the error pattern introduced by the BSC with cross-over probability $\alpha$ such that $r = x \oplus e$, and $\oplus$ is the component-wise modulo-two sum. The support of an error vector $e = (e_1, e_2, \ldots, e_N)$, denoted by supp$(e)$, is defined as the set of all positions $i$ such that $e_i \neq 0$. Let $y = (y_1, y_2, \ldots, y_N)$ be the input to the decoder, where each $y_i$ is calculated based on the received value $r_i$, and is referred to as channel value. During the analysis of decoders, we shall assume that the all-zero codeword was transmitted. This is a valid assumption since the decoders we consider are symmetric [8].

A trapping set (TS) [3] for an iterative decoder denoted by $T$ is a non-empty set of variable nodes in $G$ that are not correct at the end of a given number of iterations. Note
that $T$ will depend on the decoder input as well as decoder implementation. A common notation used to denote a TS is $(a, b)$, where $a = |T|$, and $b$ is the number of odd-degree check nodes in the subgraph induced by $T$.

Let $T(a, b)$ denote the bipartite graph associated with an $(a, b)$ TS, where $a$ is the number of variable nodes and $b$ is the number of odd-degree check nodes present in the graph. A graph $G$ contains an $(a, b)$ TS of type $T$ if there exists a subset of variable nodes $T$ in $G$ whose induced subgraph is isomorphic to $T(a, b)$. A TS is said to be elementary if $T$ contains only degree-one and/or degree-two check nodes. Throughout this paper, we restrict our focus to elementary trapping sets, since they are known to be dominant in the error floor [3], [29]. Henceforth, for convenience, whenever we refer to a TS, we will implicitly refer to its topological structure $T$.

IV. FINITE ALPHABET ITERATIVE DECODERS

We shall now introduce a new type of finite precision decoder which we refer to as a FAID [13], [14]. An $N_s$-level FAID denoted by D is a 4-tuple given by $D = (M, \mathcal{Y}, \Phi_v, \Phi_c)$. The finite alphabet $M$ defined as $M = \{-L_0, \ldots, -L_2, -L_1, 0, L_1, L_2, \ldots, L_s\}$, where $L_i \in \mathbb{R}^+$ (set of strictly positive real numbers) and $L_i > L_j$ for any $i > j$, consists of $N_s = 2s + 1$ levels to which the message values are confined to. The sign of a message $x \in M$ can be interpreted as an estimate of the bit associated with the variable node to or from which $x$ is being passed (positive for zero and negative for one), and the magnitude $|x|$ as a measure of how reliable this value is.

The set $\mathcal{Y}$ denotes the set of all possible channel values. For FAIDs over the BSC, $\mathcal{Y}$ is defined as $\mathcal{Y} = \{\pm C\}$, where $C \in \mathbb{R}^+$, and the value $y_i \in \mathcal{Y}$ for node $v_i$ is determined by $y_i = (-1)^{v_i}C$, i.e., we use the mapping $0 \rightarrow C$ and $1 \rightarrow -C$. Let $m_1, \ldots, m_{d_v-1}$ denote the extrinsic incoming messages to a node with degree $d_v$.

A. Definitions of the Update Functions $\Phi_v$ and $\Phi_c$

The function $\Phi_c : M^{d_{v}-1} \rightarrow M$ used for updating a check node with degree $d_v$ is defined as

$$\Phi_c(m_1, \ldots, m_{d_v-1}) = \left( \prod_{j=1}^{d_v-1} \text{sgn}(m_j) \right) \min_{j \in \{1, \ldots, d_v-1\}} (|m_j|)$$

where $\text{sgn}$ is the signum function. Note that $\Phi_c$ is the same in all $N_s$-level FAIDs, and it is the same function used in the min-sum decoder. Hence, the novelty in the proposed decoders lies in the definition of the function $\Phi_v$, which uniquely defines an $N_s$-level FAID.

The function $\Phi_v : \mathcal{Y} \times M^{d_{v}-1} \rightarrow M$ used for update at a variable node with degree $d_v$ is defined in closed form as

$$\Phi_v(y_i, m_1, m_2, \ldots, m_{d_v-1}) = Q \left( \sum_{j=1}^{d_v-1} m_j + \omega_i \cdot y_i \right)$$

where the function $Q(.)$ is defined below on a threshold set $\mathcal{T} = \{T_i : 1 \leq i \leq s + 1\}$ with $T_i \in \mathbb{R}^+$ and $T_i > T_j$ if $i > j$, and $T_{s+1} = \infty$.

$$Q(x) = \begin{cases} \text{sgn}(x)L_i, & \text{if } T_i \leq |x| < T_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

The weight $\omega_i$ is computed using a symmetric function $\Omega : M^{d_{v}-1} \rightarrow \mathbb{R}^{\geq 0}$ (set of non-negative real numbers) whose input arguments are the $d_v - 1$ incoming messages. Note that $\Omega$ is invariant to the ordering of its arguments. Based on this definition, the function $\Phi_v$ can be classified as a linear-threshold (LT) function or a non-linear-threshold (NLT) function. If $\Omega = 1$ (or constant), i.e., if the value of $\omega_i$ is always 1 (or constant) for all possible inputs of $\Omega$, then $\Phi_v$ is an LT function and a FAID with such a $\Phi_v$ is classified as an LT FAID. Else, $\Phi_v$ is an NLT function and a FAID with such a $\Phi_v$ is an NLT FAID.

Note that for an LT FAID, $\Phi_v$ takes a linear combination of its arguments and then applies the function $Q$ to determine its output. Therefore, $\Phi_v$ will always output the same value for any possible set of incoming messages for a given $y_i$, if their sum remains the same. For example, for a node with $d_v = 3$, $\Phi_v(-C, m_1, m_2) = \Phi_v(-C, m_3, m_4)$ when $m_1 + m_2 = m_3 + m_4$. This is also a typical property present in existing quantized decoders such as quantized BP and min-sum.

On the other hand, for an NLT FAID, $\Phi_v$ takes a non-linear combination of its arguments (due to $\Omega$) before applying the function $Q$ on the result. Therefore, $\Phi_v$ can output different values even for distinct sets of incoming messages that have the same sum. For instance, consider a map $\Phi_v$ for a node with $d_v = 3$ such that $\Phi_v(-C, -L_3, L_3) = 0$ and $\Phi_v(-C, -L_3, -L_2) = -L_3$. In this case, the two distinct sets of incoming messages are $\{-L_3, L_3\}$ and $\{-L_3, -L_2\}$, and the sums are zero for both the sets. However, $\Phi_v$ still gives different outputs for each of the sets namely, 0 and $-L_1$ respectively. Hence these decoders are different from existing quantized message-passing decoders.

Note that both update functions $\Phi_v$ and $\Phi_c$ are symmetric on the incoming messages, i.e., they remain unchanged by any permutation of its messages. Furthermore, for FAIDs considered in this paper, the function $\Phi_v$ must satisfy the following two properties.

Property 1 (Property of symmetry).

$$\Phi_v(y_i, m_1, \ldots, m_{d_v-1}) = -\Phi_v(-y_i, -m_1, \ldots, -m_{d_v-1})$$

Property 2 (Property of monotonicity).

$$\Phi_v(y_i, m_1, \ldots, m_{d_v-1}) \geq \Phi_v(y_i', m_1', \ldots, m_{d_v-1}') \text{ when } m_j \geq m_j' \forall j \in \{1, \ldots, d_v - 1\}.$$

Property 2 ensures that for a given channel value, the output is always non-decreasing with increase in the values of the incoming messages. For instance, a map $\Phi_v$ where $\Phi_v(-C, L_1, L_2) = L_2$ and $\Phi_v(-C, L_2, L_2) = L_1$ is forbidden, since $\Phi_v(-C, L_2, L_2)$ must not be less than $\Phi_v(-C, L_1, L_2)$. This is also a typical property found in existing message-passing decoders.

Note that Property 2 is enforced on $\Phi_v$ only with respect to the arguments involving the incoming messages, and not the channel value. In other words, $\Phi_v$ does not need to satisfy the constraint $\Phi_v(C, m_1, \ldots, m_{d_v-1}) \geq$
Φ_v(−C, m_1, ..., m_k, −1). However, as expected, we found that FAIDs whose Φ_v is not monotonic on the channel value argument exhibit very poor DE thresholds and therefore poor error-rate performance. The Φ_v for FAIDs obtained from the proposed methodology satisfy the monotonicity on the channel value argument as well.

At the end of each iteration, the bit value corresponding to each node v_i is determined based on the sign of the sum of all incoming messages plus the channel value y_i (zero if sign is positive, one if sign is negative, and r_i if sum is zero).

It is evident from the definition that Φ_v can be uniquely described either by assigning real values to the elements of M, Ψ and Υ, and defining Ω, or by providing a set of constraints which the assigned values can take. As examples, we provide the closed-form description of Φ_v for a 5-level NLT FAID and a 7-level LT FAID defined for column-weight-three codes.

**Example 1 (5-level NLT FAID).** The constraints on the values assigned to elements of M and Ψ that describe this map are: C = L_1, L_2 = 3L_1, T_1 = L_1, T_2 = L_2, and the function Ω is given by w_i = Ω(m_1, m_2) = 1 − (U(m_1) ∪ U(m_2)) − δ(|m_1| + |m_2| − 2L_2), where U is the unit step function and δ is the Kronecker delta function.

**Example 2 (7-level LT FAID).** The constraints on the values assigned to elements of M and Ψ that describe this map are: L_1 < C < 2L_1, L_2 = 2L_1, L_3 = 2L_2 + C, and T_1 = L_1, T_2 = L_2, and T_3 = L_3 − C, where Ω = 1.

**B. Array Representation of Φ_v**

The function Φ_v can alternatively be defined using d_v−1-dimensional arrays or look-up tables (LUTs) rather than as closed-form functions, which enables simple implementations and also may be more convenient for decoder selection. Let M be defined as M = {M_1, M_2, ..., M_N}, where M_1 = L_1, M_2 = L_1, M_3 = 2L_2 + C, and T_1 = L_1, T_2 = L_2, and T_3 = L_3 − C, where Ω = 1.

**TABLE I**

<table>
<thead>
<tr>
<th>m_1/m_2</th>
<th>L_3</th>
<th>L_2</th>
<th>L_1</th>
<th>0</th>
<th>+L_1</th>
<th>+L_2</th>
<th>+L_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>−L_3</td>
<td>l_1, i</td>
<td>l_2, i</td>
<td>l_3, i</td>
<td>l_4, i</td>
<td>l_5, i</td>
<td>l_6, i</td>
<td>l_7, i</td>
</tr>
<tr>
<td>−L_2</td>
<td>l_2, i</td>
<td>l_2, i</td>
<td>l_3, i</td>
<td>l_4, i</td>
<td>l_5, i</td>
<td>l_6, i</td>
<td>l_7, i</td>
</tr>
<tr>
<td>−L_1</td>
<td>l_3, i</td>
<td>l_4, i</td>
<td>l_5, i</td>
<td>l_6, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
</tr>
<tr>
<td>0</td>
<td>l_4, i</td>
<td>l_4, i</td>
<td>l_5, i</td>
<td>l_5, i</td>
<td>l_6, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
</tr>
<tr>
<td>+L_1</td>
<td>l_5, i</td>
<td>l_5, i</td>
<td>l_5, i</td>
<td>l_5, i</td>
<td>l_6, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
</tr>
<tr>
<td>+L_2</td>
<td>l_6, i</td>
<td>l_6, i</td>
<td>l_6, i</td>
<td>l_6, i</td>
<td>l_6, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
</tr>
<tr>
<td>+L_3</td>
<td>l_7, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
<td>l_7, i</td>
</tr>
</tbody>
</table>

It is easy to see that if the diagonal entries in the array \([l_{i,j}]\)_{1\leq i\leq N, 1\leq j\leq N} are different, then Φ_v must be an NLT function (as discussed previously). However, Φ_v can still be an NLT function even if all the diagonal entries in the array are the same, as highlighted in the following lemma.

**Lemma 1.** If Φ_v(−C, −L_1, L_2) = −L_1, Φ_v(−C, 0, L_1) = 0, Φ_v(−C, 0, −L_2) = −L_2, and Φ_v(−C, −L_1, −L_1) = −L_2, then Φ_v cannot be expressed as a linear-threshold function, and is therefore a non-linear-threshold function.

**Proof:** Assume that there exists an LT representation for such a Φ_v, which is defined by assigning real values to the elements of the alphabet M, the threshold set Ψ, and the set Υ. Since Φ_v(−C, −L_1, L_2) = −L_1, the inequality L_1 − L_1 + C ≥ T_1 must hold. Also since Φ_v(−C, 0, L_1) = 0, we have |C − L_1| < T_1. Combining the two inequalities, we get L_1 − L_1 + C > |C − L_1|. Adding L_1 − C on both sides, we have 2L_1 − 2L_2 > |L_1 − C| = 2L_1 + C ≥ 2L_1 > L_2. But since Φ_v(−C, 0, −L_2) = −L_2 and Φ_v(−C, −L_1, −L_1) = −L_2, we have L_2 + C > 2L_1 + C ≥ 2L_1 > L_1, which is a contradiction.

Consequently, the map defined by Table II is an NLT function. Fig. 1 shows the frame error-rate (FER) performances of the 5-level NLT and 7-level LT FAIDs defined in the two examples, and the 7-level NLT FAID defined by the Table II, along with the floating-point BP and min-sum decoders on the well-known (155, 64) Tanner code [30]. The 5-level NLT and 7-level LT FAIDs were designed heuristically for the Tanner code while the 7-level NLT FAID (denoted as D_7) was designed using our proposed selection methodology. All decoders were run for a maximum of 100 iterations. From the plot, we see that all the FAIDs significantly outperform the floating-point BP and min-sum on the code. The methodology used to select FAIDs will be described shortly.

Due to properties 1 and 2 of Φ_v, there exists a useful connection between FAIDs, and symmetric plane partitions which are well-known tools used in combinatorics. A symmetric plane partition π is an array of non-negative integers (π_{i,j})_{1\leq i,j\leq 1} such that π_{i,j} ≥ π_{i+1,j}, π_{i,j} ≥ π_{i,j+1} for all i, j ≥ 1, and π_{i,j} = π_{j,i}. If π_{i,j} = 0 for all i, j ≥ 1, then the plane partition is said to be contained in a box of the form \([r \times s \times t]\). The value π_{i,j} is represented as a box of height \(h_{i,j}\) positioned at the \((i,j)\) coordinate on a horizontal plane. A bijection can be constructed between the set of N_1-level FAIDs and the set of symmetric plane partitions contained in a box of the form \([N_1 \times N_s \times N_s − 1]\) by assigning the value for π_{i,j} as the \(i,j\)-th entry of an \([N_1 \times N_s \times N_s − 1]\) matrix.
corresponding to $\Phi_v$ of the 7-level FAID $D_0$ defined by Table II.

Kuperberg in [31] gave an elegant formula for the enumeration of symmetric plane partitions contained in a box, and we can directly utilize this for the enumeration of $N_v$-level FAIDs as well [13]. Using this formula, the total number of FAIDs for $N_v = 5$ and $N_v = 7$ levels are 28,314 and 530,803,988 respectively. Clearly, identifying particularly good FAIDs from the set of all possible $N_v$-level FAIDs (even for $N_v = 7$) is highly non-trivial due to its large cardinality.

V. SELECTION OF FINITE ALPHABET ITERATIVE DECODERS

We now describe a practical approach to identify a subset of $N_v$-level FAIDs, one or several among which are potentially good for a given column-weight-three code. Our main aim behind this approach is to restrict the choice of FAIDs to a possibly small subset containing good candidates. Given a particular code, it would then be feasible to identify the best performing FAID from this smaller subset by exhaustively checking each one through simulation. Moreover, since the performance of a FAID on a particular code actually depends on its Tanner graph, a FAID that provides exceptional FER performance on one code may not necessarily perform as well on another code. Therefore, the goal of identifying several candidate FAIDs is more realistic than identifying a single good FAID and allows for devising a selection method that is not code-specific. Another important objective of our approach is to ensure that any FAID belonging to this subset is capable of surpassing BP in the error floor not just on a single code, but on several codes.

The approach we use relies on the knowledge of potentially harmful subgraphs that could be trapping sets for traditional iterative decoders when present in a given code. The candidate FAIDs are chosen by analyzing their behavior on each of these subgraphs with errors introduced in them. We will first introduce some important notions that form the basis of our approach, and then subsequently present a methodology for FAID selection for column-weight-three codes.

A. Critical Number and Isolation Assumption

The notion of critical number associated with a TS was originally introduced for Gallager-A/B algorithms on the BSC [29], [32] and its definition is provided below.

Definition 1. The critical number of $T(a,b)$ is the minimum number of variable nodes in $T$ that have to be initially in error for the Gallager-A/B decoder to fail.

The critical number provides a measure of how harmful a TS is when present in a graph $G$, as the slope of the error floor in the FER curve is most affected by the uncorrectable error patterns of smallest weight [5]. Hence, this notion is useful for predicting the error floor performance [29] as well as for determining the harmful subgraphs that should be avoided in the code designs [6]. From a decoder design perspective, it would be useful to extend this notion to FAIDs so that the harmfulness of a TS could be studied under different FAIDs.

In order to enable this extension, we require the notion of isolation assumption [34], which is a condition assumed on the graph $G$ containing the subgraph $T(a,b)$. Under this assumption, the neighborhood of the TS is such that the messages flowing into the TS from its neighborhood are not in any way influenced by the messages flowing out of the TS. Therefore, the messages flowing into the TS can be computed while completely disregarding the neighborhood [34, Theorem 1], and $T(a,b)$ can be analyzed under a particular FAID in an isolated manner. We shall now precisely define this notion.

Let $\mathcal{F}_k^k(G)$ denote the computation tree corresponding to an iterative decoder on $G$ enumerated for $k$ iterations with node $v_i \in V$ as its root. A node $w \in \mathcal{F}_k^k(G)$ is a descendant of a node $u \in \mathcal{F}_k^k(G)$ if there exists a path starting from node $w$ to root $v_i$ that traverses through node $u$.

Definition 2 (Isolation assumption). Let $H$ be a subgraph of $G$ induced by $P \subseteq V$ with check node set $W \subseteq C$. The computation tree $\mathcal{F}_k^k(G)$ with the root $v_i \in P$ is said to be isolated if for any node $u \notin P \cup W$ in $\mathcal{F}_k^k(G)$, $u$ does not have any descendant belonging to $P \cup W$. If $\mathcal{F}_k^k(G)$ is isolated

Fig. 1. Performance comparisons between the floating-point decoders: BP and min-sum , and the 3-bit precision decoders: 5-level NLT, 7-level LT, and 7-level NLT (D_0) FAIDs on the (155, 64) Tanner code.

Fig. 2. A visualization of the plane partition as stacked boxes for the 7-level FAID whose $\Phi_v$ is described in Table II.
∀v ∈ P, then H is said to satisfy the isolation assumption in G for k iterations.

The above definition is a revised version of the one given in [34]. Note that for a graph G with girth 8, the isolation assumption will be satisfied for at least two iterations. Also note that it is weaker than Gallager’s independence assumption as explained in [34]. The critical number can now be extended for FAIDs.

**Definition 3.** The critical number of a FAID D on T(a, b) is the minimum number of variable nodes in T that have to be initially in error for D to fail under the isolation assumption.

We set the critical number to ∞ if D corrects all possible error patterns in T(a, b). The critical number can now be used as a possible parameter for decoder selection where a decoder is chosen to maximize the critical number on a given number of TSs. In principle, one could consider a database of potential TSs that are generated either through analytical or empirical evaluations of traditional decoders such as BP and min-sum on several different codes, and then select a FAID based on its critical numbers on all these TSs.

However, the isolation assumption of a TS typically does not hold in an actual code for more than few iterations, and the influence of the neighborhood can play a greater role especially when the TSs considered are of small size. Hence, the critical number may not reflect the true error-correction capability of the FAID on a code containing the TS. Unless a very large database of TSs oe TSs with large sizes are considered (as done in [35] where the considered TSs are the weight-20 codewords), the strategy will remain ineffective. This motivates the need for a new notion that considers, to an extent, the influence of the neighborhood.

**B. Noisy Trapping Sets and Noisy Critical Numbers**

Consider the analysis of a FAID D on T(a, b) with errors introduced in T (by setting variable nodes in T to be initially in error). In order to at least partially capture the influence of an arbitrary (unknown) neighborhood of T, we do the following. We first carry out message-passing on T under the isolation assumption for only two iterations (which is satisfied since we focus on girth-8 graphs). Under this assumption, the message passed by the degree-one check node from the a,b node is corrected by the FAID within N1 iterations under a particular FAID D on a T(a, b).

**Definition 4.** An initialization vector on T(a, b) is defined as a vector Θ = (θ1, ..., θb) where θi ∈ M, such that during the message-passing of a FAID on T(a, b), θi is the message passed by the ith degree-one check node from the third iteration up to N1 iterations. The TS T(a, b) is said to be initialized by such a vector and is referred to as a noisy trapping set.

Note that T(a, b) is initialized by vector Θ only after two iterations of message-passing under the isolation assumption, and it is carried out only through the degree-one check nodes. Also note that the initialization vector Θ is not iteration-dependent.

As an example, Fig. 3 depicts how a FAID is analyzed for a three-error pattern in a T(6, 2) initialized by a vector Θ = (θ1, θ2) in the third iteration of message-passing. A • denotes a variable node initially in error (v1, v2, and v4) and a ○ denotes a node initially correct (v5, and v6). A □ denotes a degree-two check node and a ■ denotes a degree-one check node. Initially all messages are set to zero. Then, the messages are iteratively updated using the maps Φ1 and Φ2 by treating the subgraph T in an isolated manner as if it were the Tanner graph of a code but with the exception that a degree-one check node sends to its neighbor μ = Φv(C, 0, 0) in the first iteration, Φv(C, μ, μ) in the second iteration, and then θ1 (or θ2) for the remaining iterations. The message update on a single edge in the third iteration from a variable node is shown for each of the nodes v1, v2, v3, and v5 (v4 and v6 are similar to v2 and v3 respectively). Note that the messages m1, m2, ..., m6 denote the extrinsic incoming messages to these nodes.

![Fig. 3. An example of a noisy T(6, 2) initialized by a vector Θ in the third iteration.](image-url)
after the second one, and different initializations represent the different possible influences that the neighborhood of \( H \) can have. Therefore, analyzing the FAID under different initializations on \( T(a,b) \) can provide a good indication of its error correction capability on a code whose graph \( G \) contains \( H \).

Although the initialization vector should ideally be iteration-dependent and include all messages passed to all check nodes of \( T(a,b) \) from outside of \( T(a,b) \), this would make analyzing a FAID on \( T(a,b) \) computationally intractable. We therefore only include constant values that are passed by degree-one check nodes in the initialization vector. Let \( \{ \Theta_1, \Theta_2, \ldots, \Theta_{N_\Theta} \} \) denote the set of all possible initialization vectors on \( T(a,b) \), where \( N_\Theta = |M^b| \). We now define the notion of noisy critical number vector for FAIDs.

**Definition 5.** The noisy critical number vector (NCNV) of a FAID \( D \) on \( T(a,b) \) is defined as \( \Phi(D, N_1) = (\xi_1, \xi_2, \ldots, \xi_{N_\Theta}) \), where \( \xi_i \) is the minimum number of variable nodes that have to be initially in error for \( D \) to fail after \( N_1 \) iterations under initialization vector \( \Theta_i \in M^b \).

The NCNV of a FAID can now be computed for different TSs (note: for ensuring that all NCNVs have the same dimension, only TSs with same \( b \) are considered). However, in order to be used as a parameter for decoder selection, we need to be able to compare the NCNVs of different FAIDs in an effective way. We achieve this through the notion of decoder domination.

**C. Decoder Domination**

Let \( \mathcal{F} = \{D_1, \ldots, D_{N_\mathcal{F}} \} \) denote the set of \( N_\mathcal{F} \)-level FAIDs considered for decoder selection, and let \( \Lambda = \{ T_1, \ldots, T_{N_\Lambda} \} \) denote the set of TSs used for selecting FAIDs from \( \mathcal{F} \). An NCNV of FAID \( D_b \in \mathcal{F} \) can be computed on every \( T_j \in \Lambda \). Suppose we want to compare these NCNVs with the NCNVs of another FAID \( D_l \in \mathcal{F} \). To facilitate this comparison conveniently, an array \( A_{D_b} \) of size \( N_\Theta \times N_\Lambda \) can be defined whose \( j^{th} \) column corresponds to the NCNV computed on \( T_j \). In other words, the entry \( A_{D_b}(i,j) \) is the noisy critical number of \( D_b \) computed on \( T_j \) under initialization vector \( \Theta_i \).

We now compare the two arrays \( A_{D_b} \) and \( A_{D_l} \) using the notion of domination multiplicity. The domination multiplicity of \( D_b \) on \( D_l \) denoted by \( \tilde{n}(D_b, D_l) \) is given by

\[
\tilde{n}(D_b, D_l) = \sum_{i=1}^{N_\Theta} \left( \prod_{j=1}^{N_\Lambda} \mathbb{I}(A_{D_b}(i,j) \geq A_{D_l}(i,j)) \right) W_{D_b,i}
\]

(3)

where \( \mathbb{I} \) is the indicator function that outputs a one when the condition in its argument is true, and zero otherwise, and \( W_{D_b,i} \) is a non-negative weight corresponding to a vector \( \Theta_i \) that depends on \( D_b \). Plainly speaking, the domination multiplicity of \( D_b \) on \( D_l \) is computed by identifying a vector \( \Theta_i \) for which \( u_{D_b}(T_j, N_j) \geq u_{D_l}(T_j, N_j) \) \( \forall T_j \in \Lambda \), and then taking a weighted sum of all such vectors.

The weight \( W_{D_b,i} \) is used to signify the fact that certain initialization vectors may be more important than others in terms of a FAID having a higher critical number, since for a particular FAID, certain vectors may be more likely to occur than others during actual message-passing on a graph \( G \). In other words, the weight \( W_{D_b,i} \) gives a measure of how likely the vector \( \Theta_i \) is to occur during decoding by the FAID \( D_b \). We capture this measure by choosing these weights in the following manner. Let \( \Phi(D_b) \) denote the map of FAID \( D_b \). For a component of the vector \( \Theta_i \), say \( \theta_p \), we obtain a multiplicity by counting the number of unordered pairs of messages \((m_1, m_2) \in M^2 \) for which \( \Phi(D_b)(C, m_1, m_2) = \theta_p \). The weight \( W_{D_b,i} \) is then the product of all the multiplicities obtained for each component. For ease of understanding, let us consider the case of \( b = 2 \). Then \( W_{D_b,i} \) for a vector \( \Theta_i = \{\theta_1, \theta_2\} \) is precisely given by

\[
W_{D_b,i} = \left\{ \left( (m_1, m_2), (m_3, m_4) \in M^2 : \Phi(D_b)(C, m_1, m_2) = \theta_1, \Phi(D_b)(C, m_3, m_4) = \theta_2 \right) \right\}
\]

where \((m_1, m_2) \) and \((m_3, m_4) \) are unordered pairs of messages. If a vector \( \Theta_i \) has a higher associated weight than a vector \( \Theta_p \), we regard \( \Theta_i \) to be more likely to occur since it has a greater number of possible entries in \( \Phi(D) \), that output its corresponding components. We found this choice of weights to be appropriate for the decoder selection. Note that for any \( N_\mathcal{F} \)-level FAID \( D_b \), the weights \( \{W_{D_b,i}\}_i \) will satisfy

\[
\sum_{i=1}^{N_\Theta} W_{D_b,i} = \left( \left( \frac{N_\mathcal{F}}{2} \right) + N_\Lambda \right)^b
\]

If \( \tilde{n}(D_b, D_l) > \tilde{n}(D_l, D_b) \), then \( D_b \) is said to dominate \( D_l \) with domination strength \( \tilde{n}(D_b, D_l) - \tilde{n}(D_l, D_b) \). A high domination strength implies that there is a much larger number of vectors (or vectors that have higher weights) for which \( D_b \) has higher critical numbers than \( D_l \). For convenience, we shall use the symbol \( \triangleright \) to denote domination, i.e., \( D_b \triangleright D_l \) implies that \( D_b \) “dominates” \( D_l \).

**D. Choice of Trapping Sets for Decoder Selection: Using the Trapping Set Ontology**

Since our approach for identifying good FAIDs relies on comparing the NCNVs of FAIDs computed on different TSs, the choice of TSs to be included in the set \( \Lambda \) plays an important role in the effectiveness of the decoder selection. We utilize the trapping set ontology (TSO) [33] of column-weight-three codes in order to choose the TSs. The TSO is a database of TSs that is organized as a hierarchy based on their topological relations specified in the form of a predecessor-successor relationship [6]. A trapping set \( T_1 \) in the TSO is said to be a predecessor of a trapping set \( T_2 \) if \( T_2 \) contains \( T_1 \), and \( T_2 \) is considered to be a successor of \( T_1 \). The TSO is generated based on the trapping sets for Gallager-B which are characterized combinatorially using the theorem provided in [32]. This is motivated by the observation from Chiellappagari et al. that trapping sets for various decoders over different channels are closely related [37].

As an illustration, Fig. 4 provides a partial list of graphs belonging to the TSO of girth-8 column-weight-three codes which originates from an 8-cycle as a \((4,4) \) TS, and moves right towards larger graphs by adding extra nodes. The
predecessor-successor relationship between the graphs is highlighted through the arrows directed from the predecessors towards successors. Note that \((a, b)[i]\) is used to distinguish between multiple graphs in the TSO having the same \((a, b)\). Refer to [38] for a more exhaustive list of TSs in the TSO.

In order to select graphs from the TSO to be included in set \(\Lambda\), we use the following criteria: 1) we select graphs from the TSO which together cover as many of the harmful predecessors in the TSO as possible but also with each graph having some successors in the TSO not considered, and 2) the graphs selected must ensure reasonable complexity in the computation of the NCNVs since larger graphs will require more computations. The main intuition for criterion 1 is that the successors are as harmful or more than their predecessors since they contain their predecessors. On the other hand, selecting only the successors with the least values of \(b\) such as the \((7,1)\) TS may render the selection method ineffective, since even the good FAIDs may not have a good error correction capability on them. Further, the harmfulness of predecessors can be known based on failures of conventional decoders. For example, the \(T(5,3)\) TS has been found to be more harmful than the \(T(6,4)\{1\}\) TS for Gallager-B [29] as well as BP decoding [6], so successors of \(T(5,3)\) TS should be chosen. A more elaborate discussion on the harmfulness of TSs can be found in [6]. An additional criterion that can be used is to take into account graphs that are more commonly found in practical high-rate codes.

Note that for practicality of comparing NCNVs computed on different TSs, we select graphs that have the same values of \(b\) so that the NCNVs all have the same dimension. Fixing the value of \(b\) in the choice of TSs does not diminish the effectiveness of the decoder selection, since they will contain predecessors in the TSO that can have different values of \(b\).

E. Methodology for Selection: A General Approach

We now present a methodology for identifying good \(N_s\)-level FAIDs based on the notions of decoder domination and the NCNVs. We remind the reader that the main goal of our approach is to be able to identify a possibly small subset of candidate FAIDs \(\mathcal{F}_c\) from the set of \(N_s\)-level FAIDs \(\mathcal{F}\), where each candidate FAID in \(\mathcal{F}_c\) is potentially good on several codes. Ideally, if a candidate FAID could be selected solely based on how it dominates all the other FAIDs in \(\mathcal{F}\), then one could possibly obtain an ordering of the FAIDs in \(\mathcal{F}\) in terms of their dominance and conclude as to which ones are more likely to be good on a given code. Unfortunately, we have found that such an ordering does not exist since there can be many FAIDs that dominate a particularly good FAID (known a priori to be good) and yet perform poorly on certain codes.

Therefore, we propose an approach for selection that utilizes pre-determined small sets of “good” FAIDs and “bad” FAIDs denoted by \(\mathcal{F}_g\) and \(\mathcal{F}_b\) respectively. Let a FAID in \(\mathcal{F}_g\) be denoted by \(G_i\), and a FAID in \(\mathcal{F}_b\) be denoted by \(B_i\) (\(i\) being the index number). A “good” FAID \(G_i\) is known a priori to have good FER performance (surpassing BP) on several codes, whereas a “bad” FAID \(B_i\) in \(\mathcal{F}_b\) may perform well on one code but not on several codes. We regard FAIDs in \(\mathcal{F}_b\) to be “bad” since our goal is to identify FAIDs that perform well on several codes. We shall first present the general methodology used to select FAIDs, and then later we provide the specific details used to generate the list of candidate FAIDs provided in this paper.

The selection procedure begins with computing the NCNVs on each of the TSs \(\Lambda\) for all the FAIDs contained in the sets \(\mathcal{F}_c, \mathcal{F}_g\), and \(\mathcal{F}_b\). The value of \(N_f\) used to compute the NCNVs should generally be chosen to be small (5 to 10 iterations). We then evaluate whether a particular FAID \(D_k \in \mathcal{F}\) dominates or is dominated by the FAIDs in the sets \(\mathcal{F}_g\) and \(\mathcal{F}_b\). By using the sets \(\mathcal{F}_g\) and \(\mathcal{F}_b\) to compare with, we are inherently trying to select FAIDs whose NCNVs have characteristics similar to the NCNVs of FAIDs in \(\mathcal{F}_g\) but dissimilar to the NCNVs of the FAIDs in \(\mathcal{F}_b\). Therefore, we define a cost function \(\mathcal{C}_\beta\), given below, that is based on domination strengths, and whose value determines whether the FAID \(D_k\) should be accepted for inclusion into \(\mathcal{F}_c\).

\[
\mathcal{C}_\beta(D_k) = 2 \sum_{g_i \in \mathcal{F}_g, D_k \succ B_i} (\tilde{n}(D_k, G_i) - \tilde{n}(G_i, D_k)) + \sum_{B_i \in \mathcal{F}_b, D_k \succ B_i} (\tilde{n}(D_k, B_i) - \tilde{n}(B_i, D_k)) - \sum_{g_j \in \mathcal{F}_g, g_j \succ D_k} (\tilde{n}(D_k, G_j) - \tilde{n}(G_j, D_k)) - 2 \sum_{B_j \in \mathcal{F}_b, (B_j \succ D_k)} \tilde{n}(B_j, D_k) - \tilde{n}(D_k, B_j)
\]

One crucial aspect that we have observed is that a FAID should dominate most (or all) FAIDs in \(\mathcal{F}_g\), but also not be dominated by most (or all) FAIDs in \(\mathcal{F}_b\) for it to be considered potentially good. Hence, the factors of 2 are assigned to the first and the last sum terms in \(\mathcal{C}_\beta\) to reflect this. Other factors can be chosen as long as this aspect is reflected in \(\mathcal{C}_\beta\). The value of the cost function \(\mathcal{C}_\beta(D_k)\) is then compared to a threshold \(\tau\). If \(\mathcal{C}_\beta(D_k) \geq \tau\), then the FAID \(D_k\) is selected as a candidate to be included in \(\mathcal{F}_c\), else it is rejected. The selection methodology is summarized in the form of an algorithm given below.

**Algorithm 1** FAID selection algorithm

1) Choose the parameter \(b\), and select the \((a, b)\) TSs to be included in \(\Lambda\) from the TSO. Choose the parameters \(N_f\) and \(\tau\).

2) Specify the sets \(\mathcal{F}, \mathcal{F}_g,\) and \(\mathcal{F}_b\). Initialize \(j = 1\).

3) Compute the NCNVs \(u_{G_i}(T_j, N_f) \forall G_i \in \mathcal{F}_g\) and \(u_{B_i}(T_j, N_f) \forall B_i \in \mathcal{F}_b\) on TS \(T_j \in \Lambda\). Increment \(j\) by 1 and repeat the step until \(j = N_A\). Initialize \(k = 0\).

4) Compute the NCNV \(u_{D_k}(T_j, N_f)\) of FAID \(D_k \in \mathcal{F}\) on TS \(T_j \in \Lambda\) \(\forall j \in \{1, \ldots, N_A\}\).

5) Compute the value of cost function \(\mathcal{C}_\beta(D_k)\). If \(\mathcal{C}_\beta(D_k) \geq \tau\), accept \(D_k\) into the set \(\mathcal{F}_c\). Else, reject it.

6) Increment \(k\) by 1. If \(k < N_x\), go to Step 4. Else STOP.

It is important to note that the cost function provides an ordering on the selected FAIDs, and therefore \(\tau\) only controls the cardinality of the final set \(\mathcal{F}_c\). If a large \(\mathcal{F}_c\) is desired,
then a smaller \( \tau \) is chosen, and vice versa if a smaller \( F_c \) is desired. Also note that the approach we have presented in this paper is slightly different from the one proposed in [15]. With the approach presented in this paper, we have found that the outcome of the selection procedure is greatly improved and we were able to obtain much better sets of candidate FAIDs \( F \) (in terms of their error floor performance).

Using our methodology, we were able to derive a set of good candidate 7-level FAIDs (which are 3-bit precision decoders) for column-weight-three codes. The specific parameters used in the selection algorithm to derive the set of candidate 7-level FAIDs \( F_c \) are provided below:

- In Step 1, we chose \( N_I = 6 \). Regarding the TSs, we chose \( b = 2 \) to ensure reasonable complexity and four \((a, 2)\) TSs from the TSO to be included in \( \Lambda \) based on the criteria described previously. They are \( T(6, 2) \), \( T(8, 2)\{4\} \), \( T(10, 2)\{1\} \), and \( T(10, 2)\{2\} \) whose graphs are shown in Fig. 4. Note that they are either successors of \( T(5, 3) \) or \( T(6, 4)\{2\} \). Additionally, we verified on the selected TSs that the FAIDs chosen in \( F_g \) and \( F_b \) were distinguishable in terms of their NCNVs, further validating them to be good choices. We remark that this choice of TSs may not be the only one for the selection algorithm to be effective. But with this choice, we were able to derive good candidate 7-level FAIDs.

- In Step 2, we used 12 FAIDs for the set \( F_g \) and 18 FAIDs for the set \( F_b \). The list of FAIDs used in these sets are provided concisely in Tables III and IV, where each FAID is represented by a single row specifying only the upper diagonal and diagonal entries in the LUT (Table I) of \( \Phi_\tau \). The sets were determined based on simulations on different column-weight-three test codes with different rates and lengths, one of them being the \((155, 64)\) Tanner code. The FAIDs included in \( F_g \) are the ones that were found to surpass BP on all the test codes, whereas the FAIDs in \( F_b \) were found to surpass BP only on the \((155, 64)\) Tanner code. For the set \( F_c \), instead of considering all possible 7-level FAIDs of which there is a large number, we considered a reduced set by performing density evolution and removing all the FAIDs with very poor DE thresholds.

- In Step 3, we chose \( \tau \) such that the final set \( F_c \) contained 31 candidate FAIDs. Note that \( \tau \) can be varied to obtain a larger or smaller set \( F_c \). The list of candidate 7-level FAIDs is provided concisely in Table V. Note that the LUT of FAID \( D_0 \) is also given by Table II.

On a variety of codes of different rates and lengths tested, one or several 7-level FAIDs belonging to \( F_c \) outperformed the BP (floating-point) in the error floor. Moreover, the loss in the waterfall compared to BP was found to be very reasonable. The numerical results to support this statement are provided in the next section. Another interesting feature of our selection procedure that we have found is that although the DE threshold values were not at all used as parameters in the selection of FAIDs, the candidate FAIDs that we obtained in set \( F_c \) were all found to have fairly good DE thresholds.

VI. Numerical results

In Section IV, we demonstrated the capability of 5-level and 7-level FAIDs to outperform BP in the error floor on the \((155, 64)\) Tanner code. We now provide additional numerical results on the BSC to further illustrate the efficacy of FAIDs on column-weight-three codes of higher practical interest and
### Table III
**List of 7-level FAIDs that constitute the set \( \mathcal{F}_9 \)**

<table>
<thead>
<tr>
<th>FAID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>a_1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a_2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a_3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a_4</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>a_5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a_6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a_7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Table IV
**List of 7-level FAIDs that constitute the set \( \mathcal{F}_9 \)**

<table>
<thead>
<tr>
<th>FAID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>b_0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b_1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b_2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b_3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b_4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b_5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b_6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b_7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Table V
**List of candidate 7-level FAIDs selected by the methodology that constitute the set \( \mathcal{F}_9 \)**

<table>
<thead>
<tr>
<th>FAID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>c_1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c_2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c_3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c_4</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>c_5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c_6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c_7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
validate our approach for decoder selection. The three codes used for the results in this paper were chosen to cover a broad variety of LDPC codes in terms of rate, length, and structure. They are: 1) an \( R = 0.751 \) (2388, 1793) structured code based on latin squares, 2) an \( R = 0.5 \) (504, 252) code, and 3) an \( R = 0.833 \) (5184, 4322) quasi-cyclic code. Their parity-check matrices are available in [39].

The (2388, 1793) structured code with girth-8 was designed using the method of Nguyen et al. [6], which is based on latin squares and avoids certain harmful TSs in the code design. The \( R = 0.5 \) (504, 252) code with girth-8 was constructed using the progressive edge-growth (PEG) method of [36] while ensuring that no \( T(5, 3) \) is present (but it contains \( T(6, 4) \{2 \})

The (5184, 4322) quasi-cyclic code is a high-rate girth-8 code with a minimum distance of 12.

We chose two 7-level (3-bit precision) FAIDs \( D_0 \) and \( D_1 \) in \( F_c \) (Table V) that were obtained from the methodology for making performance comparisons with BP on the three codes (choosing the best performing one out of the two). Figures 5, 6, and 7 show the FER comparisons between the 7-level FAIDs and the BP (floating-point). \( D_0 \) was used on the (2388, 1793) and the (5184, 4322) codes, while \( D_1 \) was used on the (504, 252) PEG-based code. Note that both \( D_0 \) and \( D_1 \) are 7-level NLT FAIDs. All decoders were run for a maximum of 100 iterations.

In all three codes, the 7-level FAIDs begin to surpass the BP at an FER \( \geq 10^{-5} \). Also notice the difference in the steeper slopes of the error floor for the 7-level FAIDs which can be attributed to their enhanced guaranteed error correction capability. For instance, all FAIDs used on the (155, 64) Tanner code in Fig. 1 guarantee the correction of 5 errors, whereas BP fails for several 5-error patterns that occur in the simulation. Finally, we remark that although results were provided only for three codes, we have tested the candidate 7-level FAIDs in \( F_c \) on several other codes [39], and one or several of the FAIDs in \( F_c \) outperform BP on all those codes. The results show that the proposed selection methodology, which is not code-specific, is effective in identifying potentially good 7-level FAIDs for column-weight-three codes.

\section*{VII. Conclusions}

We introduced a new paradigm for finite-precision iterative decoding of LDPC codes on the BSC. Referred to as FAIDs, the newly proposed decoders use node update maps that are much simpler than BP yet capable of surpassing the floating-point BP with only three bits of precision. We described the general framework of FAIDs with focus on column-weight-three codes and provided examples of good 3-bit precision FAIDs. We introduced the notions of NCNV and decoder domination and used them to provide a general methodology to identify a set of candidate FAIDs, one or several of which are potentially good for a given column-weight-three code. Our methodology is not code-specific, but rather utilizes the knowledge of harmful trapping sets selected from the TSO. Using this methodology, we obtained a set of potentially good
7-level FAIDs. The supporting numerical results show that by using FAIDs, it is possible to achieve a much superior error-rate performance in the error floor at a lower complexity and memory usage than BP.

ACKNOWLEDGMENT
The authors would like to thank Shashi Kiran Chilappagari for his contributions to this work.

REFERENCES

Shiva Kumar Planjery received a B.Tech in Electrical Engineering from the Indian Institute of Technology - Madras, Chennai, India, in 2005, and a M.A.Sc. in Electrical Engineering from the University of Victoria, BC, Canada, in 2007. Under a dual degree program, he received a Ph.D. from the University of Cergy-Pontoise, France, in Dec. 2012, with a specialization in communication and information theory, and a Ph.D. from the University of Arizona, Tucson, AZ, in Electrical Engineering in Aug. 2013. His current research interests include error control coding, information theory, and digital communications with specific focus on the design and analysis of low-complexity iterative decoding algorithms for LDPC codes.
David Declercq was born in June 1971. He graduated his PhD in Statistical Signal Processing 1998, from the University of Cergy-Pontoise, France. He is currently full professor at the ENSEA in Cergy-Pontoise. He is the general secretary of the National GRETSI association, and Senior member of the IEEE. He is currently the recipient of junior position at the “Institut Universitaire de France”. His research topics lie in digital communications and error-correction coding theory. He worked several years on the particular family of LDPC codes, both from the code and decoder design aspects. Since 2003, he developed a strong expertise on non-binary LDPC codes and decoders in high order Galois fields GF(q). A large part of his research projects are related to non-binary LDPC codes. He mainly investigated two aspects: (i) the design of GF(q) LDPC codes for short and moderate lengths, and (ii) the simplification of the iterative decoders for GF(q) LDPC codes with complexity/performance tradeoff constraints. David Declercq published more than 30 papers in major journals (IEEE-Trans. Commun., IEEE-Trans. Inf. Theo., Commun. Letters, EURASIP JWCN), and more than 100 papers in major conferences in Information Theory and Signal Processing.

Ludovic Danjean received the M.Sc. degrees in Electrical Engineering from the graduate school ENSEA, and the University of Cergy-Pontoise, France, in 2009. He received his Ph.D. in Electrical Engineering from the University of Cergy-Pontoise in 2012. He is currently working towards his Ph.D. in Electrical Engineering at the University of Arizona.

Bane Vasic is a Professor of Electrical and Computer Engineering and Mathematics at the University of Arizona. He is affiliated with BIO5, the Institute for Collaborative Research, and is a Director of the Error Correction Laboratory. Current sponsors and collaborators of his laboratory include NSF, DARPA, IDEMA, INSIC NASA, LANL, Seagate Technology, IBM, Hitachi, Toshiba, LSI Corp., and Bell Laboratories.

Dr. Vasic is an inventor of the soft error-event decoding algorithm, and the key architect of a decoder for Bell Labs magnetic recording read channel chips which were regarded as the best in industry. Different variants of this algorithm were implemented in virtually all magnetic hard drives. His pioneering work on structured low-density parity check (LDPC) error correcting codes and invention of codes has enabled low-complexity iterative decoder implementations. Structured LDPC codes are today adopted in a number of communications standards and in extremely high-density magnetic storage systems. He was also involved in a DVD Copy Protection Standardization Group sponsored by the DVD Industry Consortium and Hollywood Movie Studios.

Dr. Vasic is known for his theoretical work in error correction coding theory and codes on graphs which has led to analytical characterization of the hard decision iterative decoders of LDPC codes, and design of codes with best error-floor performance known today. He is an IEEE Fellow and da Vinci Fellow.