On the Selection of Finite Alphabet Iterative Decoders for LDPC codes on the BSC

Ludovic Danjean, David Declercq
ETIS
ENSEA / Univ. Cergy-Pontoise / CNRS UMR 8051
F-95000 Cergy-Pontoise, France
{danjean,declercq}@ensea.fr

Shiva K. Planjery, Bane Vasić
Dept. of Electrical and Computer Eng.
University of Arizona
Tucson, AZ 85721, USA
{shivap,vasic}@ece.arizona.edu

***Abstract*—Recently new message passing decoders for LDPC codes, called finite alphabet iterative decoders (FAIDs) were proposed. The messages belong to a finite alphabet and the update functions are simple boolean maps different from the functions used for the belied propagation (BP) decoder. The maps can be chosen using the knowledge of potential trapping sets such that the decoders surpass the BP decoder in the error floor. In this paper, we address the issue of selecting good FAIDs which perform well in the error floor for column weight three codes. We introduce the notion of *noisy trapping set* which is a generalization based on analyzing the local dynamic behaviour of a given FAID on a trapping set. Using this notion as the core, we provide an iterative greedy algorithm that outputs a set of candidate FAIDs containing potentially good decoders for any given code. To illustrate the appliance of the methodology on several codes, we show that the set of candidate FAIDs contains particularly good FAIDs for different codes with different rates and lengths.**

I. Introduction

Traditional iterative decoders for low-density parity-check (LDPC) codes are all based on the inference algorithm known as belief propagation (BP). Under BP decoding LDPC codes perform close to Shannon's theoretical limit asymptotically in the length of the code. For finite length codes however, the presence of unavoidable cycles in the Tanner graph of the code cause the BP decoder to fail on low-weight error patterns that would be correctable by the maximum likelihood decoder (MLD). This leads to an abrupt degradation in the slope of the error rate performance in the high signal-to-noise ratio (SNR) well known as the error floor region.

The error floor problem has attracted significant interest with emphasis on designing codes and low-complexity decoders with lower error floors. Richardson introduced the notion of *trapping sets* in order to characterize error floors [1]. These are certain subgraphs consisting of cycles present in the Tanner graph of the code that increase the rate of decoder failure in the high SNR region.

Recently a new class of finite alphabet iterative decoders (FAIDs) were proposed for LDPC codes over the Binary Symmetric channel (BSC), where the messages can take a finite number of levels [2]. These new decoders can provide a superior performance in the error floor compared to BP but with only a fraction of its complexity as only a small number of bits (as small as 3) are required for the message representation. The update functions used at the variable nodes are maps different from that used in BP and are chosen with the goal of increasing the guaranteed error correction capability of the code which governs the slope of the error floor on the BSC [3].

In this paper, we focus on the issue of designing good FAIDs for column weight three codes. Although the important concept of isolation assumption was introduced in [2] which allows one to analyze decoders on potential trapping sets in an isolated fashion (disregarding the neighborhood of the trapping set contained in a particular graph), a precise strategy for selecting FAIDs was not provided. In addition, on an actual code, the neighborhood can greatly influence the dynamics of message passing on the trapping set. Therefore, analysis of a FAID under isolation assumption does not accurately reflect the true decoding behaviour on a practical code, unless a much larger subgraph containing the true neighborhood (to a certain extent) is considered for the analysis. This was also identified in [4] where a code specific methodology was proposed which involved analyzing the FAIDs on subgraphs induced by the minimum-weight codewords (of size 20) and selecting them based on their error-correction capability on the subgraphs, and this gave best performing FAIDs for the code.

Our main goal in this paper is to provide a methodology that is not code specific but which gives a set of candidate FAIDs that contains potentially good decoders in the error floor for any given column-weight three code, and which utilizes only a few carefully selected small trapping sets in the analysis. To achieve this, we introduce the notion of a *noisy trapping set* which involves introducing noisy messages in the neighborhood while analyzing a given FAID on a trapping set. We then propose an iterative greedy algorithm that selects candidate FAIDs using certain statistics computed on the noisy trapping sets.

The rest of the paper is organized as follows. In Section II, we provide preliminaries required for describing FAIDs. In the Section III, we introduce the the notion of *noisy trapping sets* which forms the basis for our methodology. We then describe the proposed selection algorithm in Section IV. Finally Section V provides the results and conclusions.

II. Finite Alphabet Iterative Decoders

A. Preliminaries and notations

Let $\mathcal{C}$ denote an (N,M) binary LDPC code whose Tanner graph G has N variable nodes and M check nodes with a set of variable nodes $V = \{v_1, \cdots, v_N\}$. The degree of a node is the number of its neighbors. The code is said to be d_v-left-regular if all variable nodes in G have the same degree d_v. In this paper, we consider only 3-left-regular LDPC codes.

The ● represents a variable node, □ represents an even degree check node, and ■ represents an odd degree check node. As defined in [1], a trapping set (TS) is a non-empty set of variable nodes in G that are not eventually corrected by the decoder. A standard notation for a trapping set is (a, b) where a is the number of variable nodes and b is the number of odd-degree check nodes in the subgraph induced by the a variable nodes.

The critical number $m(a, b)$ of a TS(a, b) under a particular decoder is the minimum number of errors that need to be introduced in TS(a, b) to cause the decoder to fail on TS(a, b) under isolation assumption. More details on this notion will be discussed in the next section.

B. Definition of FAID

An N_s-level FAID $\mathscr{F}$ (as defined in [2]) is a 4-tuple given by $\mathscr{F} = (\mathcal{M}, \mathcal{Y}, \Phi_v, \Phi_c)$. The messages are levels confined to an alphabet $\mathcal{M} = \{0, \pm L_k : 1 \leq k \leq \frac{N_s - 1}{2}\}$ consisting of N_s levels, where $L_i \in \mathbb{R}^+$ and $L_i > L_j$ for any $i > j$. The set $\mathcal{Y}$ denotes the set of possible *channel values*. For the case of BSC, $\mathcal{Y} = \{\pm \mathrm{C}\}$, and for each variable node v_i in G, the channel value $y_i \in \mathcal{Y}$ is determined by $y_i = (-1)^{r_i}\mathrm{C}$ where r_i is the value received from the channel, i.e., we use the mapping $0 \to \mathrm{C}$ and $1 \to -\mathrm{C}$. We can consider the sign of $x \in \mathcal{M}$ to represent the value of the bit (postive for zero and negative for one), and the magnitude $|x|$ to reflect a measure of how reliable this value is.

$\Phi_c : \mathcal{M}^{d_c - 1} \to \mathcal{M}$ is the update function used at a check node with degree d_c. Let $m_1, \cdots, m_{d_c-1}$ denote the incoming messages used during update at a check node with degree d_c. The function is given by

$$\Phi_c(m_1, \ldots, m_{d_c-1}) = \left(\prod_{j=1}^{d_c-1} \operatorname{sign}(m_j) \right) \min_{j \in \{1, \ldots, d_c-1\}} (|m_j|)$$

where sign denotes the sign function.

$\Phi_v : \mathcal{Y} \times \mathcal{M}^{d_v - 1} \to \mathcal{M}$ is a symmetric update rule used at a variable node with degree d_v. The map Φ_v can be described in closed form as a linear or non-linear threshold function [2], or simply as a look-up table (LUT) of size $N_s^{d_v-1}$. For this paper, we shall use the LUT form. For the case of $d_v = 3$, Φ_v is a simple 2D LUT defined for each channel value.

Table I shows an example of the LUT that defines the variable node update map of a 5-level FAID for $y_i = -\mathrm{C}$ (the LUT for $y_i = +\mathrm{C}$ can be obtained by symmetry).

Since Φ_c is fixed for any N_s-level FAID, note that a particular choice of Φ_v uniquely defines the N_s-level FAID. Henceforth, for convenience we shall refer to a particular N_s-level FAID by its map Φ_v.

It is evident from the representation that there are a prohibitively large number of N_s-level FAIDs for possible decoder selection (in spite of the symmetries). Hence we further reduce the number of N_s-level FAIDs considered by specifying an additional constraint which Φ_v must satisfy as follows. For all $m_1, m_2, m_1', m_2' \in \mathcal{M}$ such that $|m_1| \geq |m'_1|$ and $|m_2| \geq |m'_2|$, Φ_v must satisfy

$$|\Phi_v(|m_1|, |m_2|, \mathrm{C})| \geq |\Phi_v(|m'_1|, |m'_2|, \mathrm{C})|$$

This constraint still allows for the selection of good FAIDs.

Table I: LUT for Φ_v of 5-level FAID with $y_i = -\mathrm{C}$

m_1/m_2	$-L_2$	$-L_1$	0	$+L_1$	$+L_2$
$-L_2$	$-L_2$	$-L_2$	$-L_2$	$-L_2$	0
$-L_1$	$-L_2$	$-L_2$	$-L_1$	$-L_1$	$+L_1$
0	$-L_2$	$-L_1$	$-L_1$	0	$+L_1$
$+L_1$	$-L_2$	$-L_1$	0	$+L_1$	$+L_2$
$+L_2$	0	$+L_1$	$+L_1$	$+L_2$	$+L_2$

Let $\mathcal{F} = \{\Phi_v^{(1)}, ..., \Phi_v^{(N_f)}\}$ denote the set of all possible variable node update maps that satisfy the above constraint, where $|\mathcal{F}| = N_f$. N_f was computed to be 28,314 for N_s = 5 and 6,392,620 for N_s = 7.

III. Noisy trapping sets

A. Inadequacy of critical numbers for decoder selection

The important notion of the critical number for a trapping set was originally introduced for Gallager-A/B algorithms on the BSC [5]. It is computed by analyzing the Gallager-A/B decoding on errors contained in the trapping set assuming that all nodes outside the trapping set are initially correct. The critical number provides a measure of how harmful a trapping set is when it is contained in a code. Hence, this notion is useful not only in evaluating the error floor performance but also in the construction of codes with lower error floor by serving as an important parameter for deciding which trapping sets to avoid during construction [6]. Alternatively, this parameter could also possibly be used in decoder designs for obtaining lower error floors where a decoder is designed or selected to increase the critical number on given trapping set(s).

In order to be able to use this notion for decoders other than Gallager-A/B such as the FAIDs and the min-sum decoder, the concept of isolation assumption was introduced in [2]. Under this assumption, the neighborhood of the trapping set is such that the messages flowing into the trapping set from its neighborhood are not in any way influenced by the messages flowing out of the trapping set. Under such a scenario, the messages flowing into trapping set can be computed while completely disregarding the neighborhood, and the decoder can be conveniently analyzed on the trapping set.

In principle, one could select a FAID (as mentioned in [2]) based on computing critical numbers on a database of potential trapping sets generated either through analytical or empirical evaluations of traditional decoders such as BP and Min-Sum on several different codes. However, the isolation assumption of a trapping set will typically not hold in an actual code for more than few iterations and hence the critical number will not reflect the true error-correction capability of the FAID on the trapping set in the code. This is especially true for trapping sets of small sizes. Therefore, unless a very large database of trapping sets with large sizes are considered such that isolation assumption holds for many more iterations, the strategy will remain ineffective. This was also shown in [4].

This motivates the need for a new notion that considers to an extent the influence of the neighborhood of a trapping set.

B. Noisy trapping sets

We now introduce the notion of *noisy trapping set*, which captures the influence of its neighborhood through the messages that are entering into the trapping set. Let N_c denote the number of checks in a TS(a,b).

***Definition* 1.** An initialization vector on TS(a,b) is defined as a vector $\mathbf{\Theta_I^{(l)}} = \{\theta_1^{(l)}, ..., \theta_{N_c}^{(l)}\}$ where $\theta_i^{(l)} \in \mathcal{M}$ such that during decoding on TS(a,b) using a particular decoder, the message entering the i^{th} check node in the l^{th} iteration is $\theta_i^{(l)}$.

***Definition* 2.** A TS(a,b) that is initialized by the initialization vector $\mathbf{\Theta_I^{(l)}}$ is called a noisy trapping set.

Our main intuition behind this notion is that different initialization vectors on the TS(a,b) mimic the different possible messages that can enter the TS(a,b) due to the influence of its neighborhood. In other words, the initialization vector on TS(a,b) in the l^{th} iteration can be considered as a possible snapshot of what the message passing looks like when decoding bits in the TS(a,b) in some arbitrary code. In this manner, we still analyze the decoder's error-correction capability on TS(a,b) but in the presence of different possible influences of its neighborhood unlike the isolation assumption case. Hence this notion of noisy trapping set gives a more accurate reflection of how the decoder would perform on TS(a,b) in an actual code. The Fig. 1 illustrates what we refer to as the full initialization of the TS(5,3).

In order to mimic all the possible messages entering TS(a,b), we would ideally need $N_s^{N_c}$ initialization vectors at each iteration, which is computationally too complex. Hence, we instead consider the initialization of a trapping set only through the odd-degree check nodes . The initialization vector is then denoted $\mathbf{\Theta_I^{(l)}} = \{\theta_1^{(l)}, ..., \theta_b^{(l)}\}$ where $\theta_i^{(l)} \in \mathcal{M}$ which reduces the number of initialization vectors. Although from the definition, an initialization vector on TS(a,b) is iteration-dependent or dynamic, we can also consider a static initialization vector that is iteration independent.

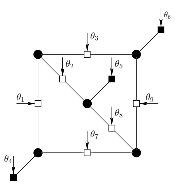

Fig. 1: Full Initialization of a TS(5,3)

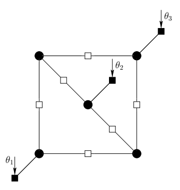

Fig. 2: Initialization of a TS(5,3)

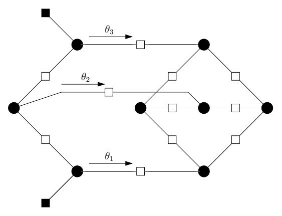

Fig. 3: TS(5,3) + 3 variable nodes = TS(8,2)

The Fig. 2 illustrates the initialization of a TS(5,3) with a static initialization vector $\mathbf{\Theta_I}$ on the odd-degree check nodes. In order to demonstrate how the initialization can mimic the influence of the neighborhood, consider the Fig. 3 where a TS(8,2) is formed by adding three extra variable nodes to TS(5,3). The initialization vector of the TS(5,3) indicates the 3 possible messages that could enter TS(5,3) due to the influence of its neighborhood defined by TS(8,2). Tables II(a) and II(b) show the evolution of messages θ_1, θ_2, and θ_3 for the first 10 decoding iterations when the 5-level FAID of Table I was employed on the (155,93) Tanner code. Table II(a) corresponds to the case when all 5 errors are introduced on a TS(5,3) contained in TS(8,2). Table II(b) corresponds to the case of adding an extra 6th error in the neighborhood of TS(5,3). From the Tables, we can see that not only does the isolation assumption not hold as messages do not remain $+L_2$ after iteration 3, but at certain iterations even messages such as 0 and $-L_1$ begin to enter the TS(5,3) due to the neighborhood.

Although we see that dynamic initialization vectors can accurately predict the decoder behaviour, we restrict ourselves to using static initialization vectors in order to ensure computational feasibility in the analysis.

Table II: Evolution of the messages entering a TS(5,3)

(a) Case 1

Iter.	$\theta_{c_1}^{(l)}$	$\theta_{c_2}^{(l)}$	$\theta_{c_3}^{(l)}$
1	$+L_1$	$+L_1$	$+L_1$
2	$+L_2$	$+L_2$	$+L_2$
3	$+L_2$	$+L_2$	$+L_2$
4	$+L_1$	$+L_1$	$+L_1$
5	0	0	0
6	$+L_1$	$+L_1$	$+L_1$
7	0	0	0
8	$+L_1$	$+L_1$	$+L_1$
9	$+L_1$	$+L_1$	$+L_1$
10	$+L_1$	$+L_1$	$+L_1$

(b) Case 2

Iter.	$\theta_{c_1}^{(l)}$	$\theta_{c_2}^{(l)}$	$\theta_{c_3}^{(l)}$
1	$+L_1$	$+L_1$	$+L_1$
2	$+L_2$	$+L_2$	$+L_1$
3	$+L_2$	$+L_1$	$+L_2$
4	$+L_2$	$-L_1$	$+L_1$
5	0	$+L_1$	0
6	$+L_2$	0	$+L_2$
7	$+L_2$	$+L_1$	$+L_1$
8	$+L_1$	0	$+L_1$
9	$+L_1$	$+L_1$	$+L_1$
10	$+L_1$	0	$-L_1$

C. Noisy critical number vector and decoder domination

Having introduced the notion of noisy trapping sets, it is natural to extend the notion of critical number as well. This can be done by computing the critical number of the TS(a,b) under a given FAID for each initialization vector that is used on TS(a,b) as opposed to computing it under isolation assumption. More precisely it is defined as follows.

Let $\mathcal{D}_i$ be an N_s-level FAID and $\Delta = \{\mathbf{\Theta}_I(k)\}_{k=1...K_v}$ be a set of K_v static initialization vectors used on a TS(a,b).

***Definition* 3.** The noisy critical number vector for a TS(a,b) under a given FAID D_i is defined as a vector

$$\mathcal{N}_{\mathcal{D}_i}\left(\mathcal{T}(a,b), \Delta\right) = \{\tilde{m}_{\mathcal{D}_i,k}\}_{k=1...Kv}$$

where $\tilde{m}_{\mathcal{D}_i,k}$ is the smallest number of errors that cannot be corrected by $\mathcal{D}_i$, when TS(a,b) is initialized using $\mathbf{\Theta}_I(k)$.

Since the noisy critical number vector $\mathcal{N}_{\mathcal{D}_i}$ reflects the error-correction capability of $\mathcal{D}_i$ on TS(a,b) under different possible influences of the neighborhood, we can use this new metric in order to select potentially good FAIDs. But before we describe the methodology, we need to introduce the concept of *domination* that allows us to compare the noisy critical number vectors between any two FAIDs in order to determine whether a FAID is potentially good or not.

For two FAIDs $\mathcal{D}_i$ and $\mathcal{D}_j$, a set of K_v initialization vectors Δ on a TS(a,b), the decoder $\mathcal{D}_i$ is said to *dominate* $\mathcal{D}_j$ if:

$$\sum_{k=1}^{K_v} \mathbb{1}\left(\tilde{m}_{\mathcal{D}_i,k} \geq \tilde{m}_{\mathcal{D}_j,k}\right) \geq \sum_{k=1}^{K_v} \mathbb{1}\left(\tilde{m}_{\mathcal{D}_i,k} \leq \tilde{m}_{\mathcal{D}_j,k}\right) \quad (1)$$

where $\mathbb{1}$ represents the indicator function of the condition in argument. In other words $\mathcal{D}_i$ will dominate $\mathcal{D}_j$ if during an elementwise comparison between the two noisy critical number vectors $\mathcal{N}_{\mathcal{D}_i}$ and $\mathcal{N}_{\mathcal{D}_j}$, the number of instances when $(\tilde{m}_{\mathcal{D}_i,k} \geq \tilde{m}_{\mathcal{D}_j,k})$ is greater than or equal to the number of instances when $(\tilde{m}_{\mathcal{D}_i,k} \leq \tilde{m}_{\mathcal{D}_j,k})$.

IV. Methodology for selection of FAIDs

We now describe a methodology that uses the notions of noisy critical number and decoder domination as the basis. Ideally, one would like to find an N_s-level FAID that dominates all other N_s-level FAIDs on a large number of potential trapping sets with reasonable values of a and b. However, we do not believe that such a decoder exists. Also from our observations, it is not possible to provide a strict ordering of the decoders in terms of how good they are for any code, since a FAID that performs the best on one code may not perform well on another code. We therefore propose a selection algorithm that provides a *collection* of several candidate FAIDs which are selected based on their analysis on noisy trapping sets, and from which one can then choose a particular FAID for any given 3-left-regular LDPC code using possibly brute force simulation of all the candidate FAIDs in the set or by some other means.

The first step is to choose the potential trapping sets and the set Δ that would be used for the analysis. Since we would like to use only a few trapping sets in the analysis for practical reasons, it is imperative that we carefully identify and select the trapping sets that are known to be the most harmful for the decoder performance. We have chosen TS(5,3), TS(7,3), TS(8,2), TS(10,2) as the noisy trapping sets based on their harmfulness and also their topological relations [6].

The next step is to construct a set of potentially "*good*" and "*bad*" decoders from the set of N_f possible FAIDs (whose maps belong to $\mathcal{F}$), namely $\mathcal{SD}(good)$ and $\mathcal{SD}(bad)$. This is done by definining an appropriate cost function and comparing its value to a threshold. Due to page limits, we have omitted details related to choice of the cost function, but basically it is based on whether a FAID D_i dominates (or is dominated by) FAIDs in $\mathcal{SD}(good)$ and $\mathcal{SD}(bad)$. Initially, $\mathcal{SD}(good)$ is empty and $\mathcal{SD}(bad)$ is initialized with a known decoder $\mathcal{D}_0$, which was chosen as the min-sum decoder. Then a FAID is chosen from the set $\mathcal{F}$, and it is either placed in $\mathcal{SD}(good)$ or $\mathcal{SD}(bad)$ depending on the value of the cost function which is compared to a threshold. This process is repeated until $\mathcal{SD}(bad)$ contains an N_B number of FAIDs. After this, the subsequent FAIDs are either placed in $\mathcal{SD}(good)$ or simply rejected or it replaces one of the FAIDs it was dominated by in $\mathcal{SD}(bad)$ with a small probability, which is all decided based on the value of the cost function. At the end, the set $\mathcal{SD}(good)$ is the final set of candidate FAIDs that are potentially good for any 3-left-regular code. The decoder selection algorithm is described below.

```
1) Define the set $\Delta$ of initialization vectors and fix
   $\mathcal{SD}(good) = \emptyset$ and $\mathcal{SD}(bad) = \mathcal{D}_0$.
2) Consider a valid FAID $\mathcal{D}_i$ whose map belongs to set $\mathcal{F}$
   and compute its noisy critical numbers
   $\mathcal{N}_{\mathcal{D}_i}(\mathcal{T}(5,3),\Delta)$, $\mathcal{N}_{\mathcal{D}_i}(\mathcal{T}(7,3),\Delta)$, $\mathcal{N}_{\mathcal{D}_i}(\mathcal{T}(8,2),\Delta)$,
   $\mathcal{N}_{\mathcal{D}_i}(\mathcal{T}(10,2),\Delta)$.
3) Define a cost function based on the domination strength
   of $\mathcal{D}_i$ with respect to the decoders in $\mathcal{SD}(good)$ and
   $\mathcal{SD}(bad)$, such that it has the following properties:
   • If $\mathcal{D}_i$ dominates $\mathcal{D}_j \in \mathcal{SD}(bad)$ and dominates
     $\mathcal{D}_m \in \mathcal{SD}(good)$, then the value of the cost
     function increases.
   • If $\mathcal{D}_i$ is dominated by $\mathcal{D}_j \in \mathcal{SD}(good)$ and is being
     dominated by $\mathcal{D}_m \in \mathcal{SD}(bad)$, then the value of
     the cost function decreases.
4) If the value of the cost function is greater than some
   threshold, place $\mathcal{D}_i$ in $\mathcal{SD}(good)$. Else
   • If $|\mathcal{SD}(bad)| < N_B$, place $D_i$ in $\mathcal{SD}(bad)$,
   • Else it means $|\mathcal{SD}(bad)| = N_B$. Then with a small
     probability replace $D_j \in \mathcal{SD}(bad)$ that dominates
     $D_i$, by $D_i$ only if the value of the cost function is
     small enough and much lower than the threshold.
     Otherwise reject it.
5) Go to 2) until all $N_f$ FAIDs have been considered.
6) $\mathcal{SD}(good)$ is the final output set of candidate FAIDs.
```
Algorithm 1: Decoder Selection Algorithm

The maximum cardinality N_B of $\mathcal{SD}(bad)$ is important to choose properly, since if N_B is too less, the role of the cost function diminishes and many FAIDs will be easily placed in $\mathcal{SD}(good)$. At the same time, it should not be too high as then it may bias the cost function to reject many good FAIDs. We chose $N_B = 20$ for our simulations and found this to be a reasonable choice. The threshold in (4) will control the cardinality of the final output set $\mathcal{SD}(good)$. This choice depends on our requirements depending on our computational resources since FAIDs need to be further chosen from the candidate set $\mathcal{SD}(good)$ for a given code. Using this methodology, we obtained for the 5-level $|\mathcal{SD}(good)| = 5$ FAIDs among 28,314 possible 5-level FAIDs tested, and $|\mathcal{SD}(good)| = 70$ 7-level FAIDs among 6,575,972 7-level FAIDs tested. The list of the final candidate FAIDs will be reported in a future publication.

V. Results and conlusions

In order to validate our approach, we selected particular FAIDs from the final candidate sets of $\mathcal{SD}(good)$ obtained for both 5 and 7 levels, for three different codes: (155,93) Tanner code, the (2640,1320) Margulis code, and a rate 0.7 (530,157) quasicyclic (QC) code. Since these codes have different lengths, rates, and different trapping set ensembles,

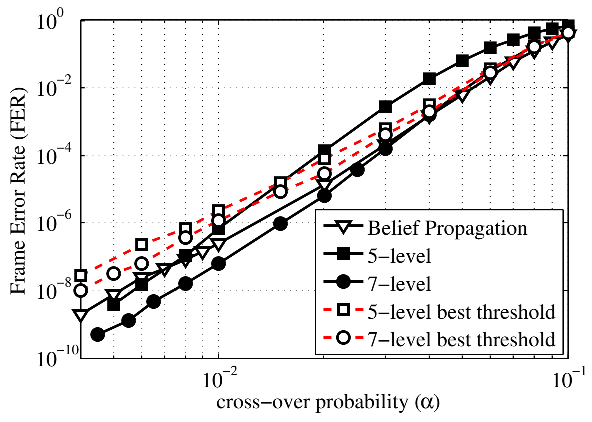

Fig. 4: FER results for the Tanner Code, N=155, R=0.413

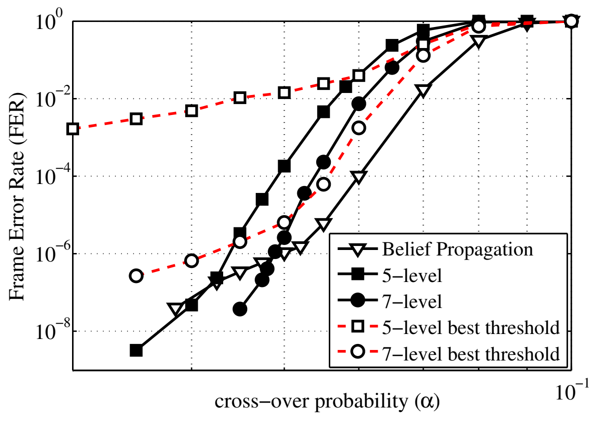

Fig. 5: FER results for Margulis LDPC code, N=2640 R=0.5

it serves as a good testbench to verify that our methodology is not code specific.

Numerical results for frame error rate vs cross-over probability (α) for different decoders on the BSC for all three codes are shown in Figs. 4, 5,6. For the Tanner code, Table I was used for the 5-level FAID, and Table III was used for the 7-level FAID. For the other two codes, the 5-level and 7-level FAIDs are different from the ones used on the Tanner code, but they all were also chosen from the candidate sets of $\mathcal{SD}(good)$. Their maps are not specified due to page constraints. For all the three codes, we see that all the 5-level and 7-level FAIDs chosen from the candidate set for each code outperform BP in the error floor. Included in the Figures are also numerical results for the FAIDs with best decoding thresholds which were computed using discrete density evolution [7]. We also see that all the 5-level FAIDs and 7-level FAIDs chosen from the candidate set outperform these FAIDs with best thresholds on all three codes, thereby showing that optimzing the decoder design for the best decoding thresholds does not necessarily lead to the best decoders. In fact the best threshold FAIDs show relatively high error foors.

We have presented a methodology that is not code specific for selecting a collection of candidate FAIDs. We remark that the procedure presented in this paper was able to provide us with a set of potentially good FAIDs fairly quickly (within 48 hours) and from which we were able to select particular FAIDs with excellent performance in the error floor for several different codes. It was also interesting to note that none of the selected candidate FAIDs had bad decoding thresholds even though density evolution was never used in the methodology.

Table III: Look-up table for Φ_v of a 7-level FAID with y_i = −C

$\boldsymbol{m_1/m_2}$	$\boldsymbol{-L_3}$	$\boldsymbol{-L_2}$	$\boldsymbol{-L_1}$	$\boldsymbol{0}$	$\boldsymbol{+L_1}$	$\boldsymbol{+L_2}$	$\boldsymbol{+L_3}$
$\boldsymbol{-L_3}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_1$
$\boldsymbol{-L_2}$	$-L_3$	$-L_3$	$-L_3$	$-L_3$	$-L_2$	$-L_1$	$+L_1$
$\boldsymbol{-L_1}$	$-L_3$	$-L_3$	$-L_2$	$-L_2$	$-L_1$	$-L_1$	$+L_1$
$\boldsymbol{0}$	$-L_2$	$-L_3$	$-L_2$	$-L_1$	0	0	$+L_1$
$\boldsymbol{+L_1}$	$-L_3$	$-L_2$	$-L_1$	0	0	$+L_1$	$+L_2$
$\boldsymbol{+L_2}$	$-L_3$	$-L_1$	$-L_1$	0	$+L_1$	$+L_1$	$+L_3$
$\boldsymbol{+L_3}$	$-L_1$	$+L_1$	$+L_1$	$+L_1$	$+L_2$	$+L_3$	$+L_3$

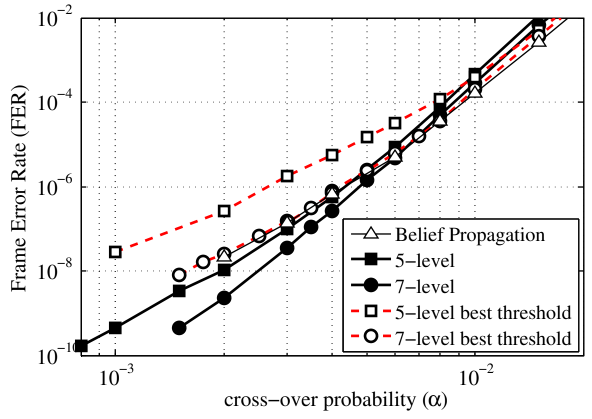

Fig. 6: FER results for QC-LDPC code, N=530 R=0.7

Acknowledgement

The authors want to thank D. V. Nguyen for providing the QC-LDPC code of length 530. This work was funded by the Nano2012 project, the French GdR-ISIS research network, and the NSF grants CCF-0830245 and CCF-0963726.

References

[1] T. Richardson, "Error Floors of LDPC Codes," in *Proc. 41st Annual Allerton Conf. on Comm. Cont. and Comput.*, 2003.

[2] S. K. Planjery, D. Declercq, S. K. Chilappagari, and B. Vasić, "Multilevel decoders surpassing belief propagation on the binary symmetric channel,"Proc. *IEEE Int. Symp. Inf. Theory*,pp.769–773,2010.

[3] M. Ivkovic, S. K. Chilappagari, and B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3763–3768, 2008.

[4] D. Declercq, L. Danjean, E. Li, S. K. Planjery, and B. Vasić, "Finite Alphabet Iterative Decoding (FAID) of the (155,64,20) Tanner Code," *6th Int. Symp. on Turbo-Codes & Iter. Inf. Proc.*, 2010, pp. 11–15.

[5] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasić, "Error floors of LDPC codes on the binary symmetric channel," in *Proc. IEEE Int. Conf. on Comm.*, vol. 3, 2006, pp. 1089–1094.

[6] B. Vasić, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology," in *Proc. 47th Annual Allerton Conf. on Comm., Cont. and Comput.*, Sept. 2009.

[7] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, 2001.