FP7-ICT / FET-OPEN – 309129 / i-RISC

D4.1

Taylor-Kuznetsov memory architectures using structured LDPC codes

Editor:	Goran Đorđević
Deliverable nature:	Public
Due date:	January 31, 2014
Delivery date:	February 3, 2014
Version	1.0
Total number of pages:	73
Reviewed by:	i-RISC members
Keywords:	LDPC codes, Taylor-Kuznetsov memory, data-dependent failures, one-step majority logic decoder, two-bit bit flipping, error-floor

Abstract

This deliverable presents an overview of the activities carried out by the work package 4 (WP4) during the first year of the project. These activities include performance evaluation of Gallager B decoding algorithm and Taylor-Kuznetsov memory architecture based on structural LDPC codes, under independent and data-dependent logic gate failures. The analytical expression for bit error rate of one-step majority logic decoders built from unreliable logic gates is derived, which enable faster analysis of regular LDPC codes compared to Monte Carlo simulation. Also, we investigate novel approaches for designing low complexity decoders and codes with good performance in error-floor region.

List of Authors

Participant	Authors
ELFAK	Goran Đorđević (goran.t.djordjevic@elfak.ni.ac.rs) Bane Vasić (vasic@ece.arizona.edu) Predrag Ivaniš (predrag.ivanis@etf.bg.ac.rs) Srđan Brkić (brka05@gmail.com)
ENSEA	David Declercq (declercq@ensea.fr) Shiva Kumar Planjery (shiva.planjery@ensea.fr)
UPT	Alexandru Amaricai (amaricai@cs.upt.ro)

Table of Contents

List of Figures

List of Tables

Abbreviations

BER	Bit Error Rate
FER	Frame Error Rate
LDPC	Low Density Parity Check
WP	Work Package
TK	Taylor-Kuznetsov
QC	Quasi-Cyclic
TBF	Two-bit Bit Flipping
EG	Euclidean Geometry
BSC	Binary Symmetric Chanel
AWGNC	Additive White Gaussian Noise Channel
SPA	Sum-Product Algorithm
TSO	Trapping Sets Ontology
UCC	University College Cork
UPT	Universitatea Politehnica Timisoara
FAID	Finite Alphabet Iterative Decoder

List of Dissemination Activities

Published papers:

1. O. Al Rasheed, S. Brkic, P. Ivanis, B. Vasic, "Performance analysis of faulty Gallager B decoding of QC-LDPC Codes," In *proceedings of 21st Telecommunication Forum, TELFOR 2013*, 26-28 november 2013, Belgrade, Serbia, pp. 323-326.
2. S. Brkic, P. Ivanis, G. Djordjevic, B. Vasic, "Taylor-Kuznetsov fault-tolerant memories: a survey and results under correlated gate failures", In *proceedings of 11th International Conference on Telecommunications in Modern Satellite and Broadcasting Services, TELSIKS 2013*, Nis, Serbia, October 16-19, 2013, pp. 455- 461.

Submitted papers:

1. S. Brkic, P. Ivanis, B. Vasic, "Analysis of one-step majority logic decoding under correlated data-dependent gate failures", submitted to *International Symposium on Information Theory (ISIT 2014)*.
2. D. V. Nguyen and B. Vasic, "Two-bit bit flipping algorithms for LDPC codes and collective error correction", submitted to *IEEE Transactions of Communications*, October 2013

Workshop presentations:

1. S. Brkic, P. Ivanis, G. Djordjevic, B. Vasic, "The analysis of Taylor-Kuznetsov fault-tolerant memories under correlated gate failures", *i-RISC Workshop presentation, European Solid-State Circuits Conference, ESSC 2013*, Bucharest Romania, September 16-19, 2013.

1. Introduction

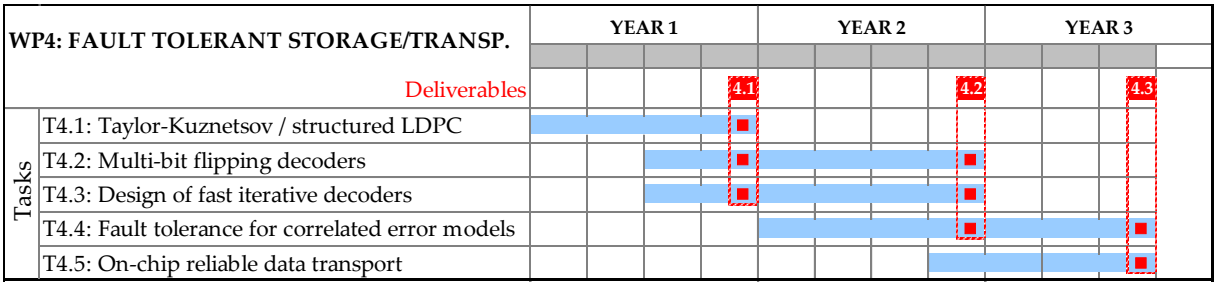
This deliverable addresses the problem of the reliable storage of digital information on a chip built from unreliable components. It represents summation of the research, done during the first year of i-RISC project, conducted in order to complete tasks described in Work Package 4 (WP4) of the project proposal. The main objectives of WP4 include analysis of the state-of-the-art memory architectures under more realistic hardware failure modeling, proposing novel memory architectures designed for tolerating both spatial and temporally correlated errors and designing the constrained codes that enable reliable intra/inter-chip bus connections. The WP4 is divided into five complementary tasks, as represented in Table 1-1, which are in the scope of our research.

In task 4.1, we investigated advanced memory architectures based on highly structured LDPC codes, namely quasi-cyclic low density parity check (QC-LDPC) codes in conjunction with the state-of-the-art iterative decoders as well as the decoders developed in the other work packages (especially WP3) with a general goal to indentify the most suitable candidate for ensuring a reliable storage. The different memory architectures are compared in terms of required sizes for a fault-tolerant memory, or equivalently the fault-tolerance level for the same size. In task 4.2 we developed fault-tolerant memories using bit parallel and serial flipping algorithms. In particular, we focused on the most recent developments on multi-bit flipping decoders, which employ additional bits at a variable node to represent its “strength”. Task 4.3 is dedicated to investigation of the error correction capabilities of finite alphabet decoders used in faulty memories. In task 4.4, based on the task 4.1 results, a novel approach to reliable memory design for correlated error models is presented. Task 4.5 contains results in designing constrained codes for robust intra/inter-chip bus communications.

Successful completion of all tasks in this WP will give us theoretical guidelines for building LDPC codes based low-complexity fault-tolerant memories and ensuring reliable date transport on unreliable hardware. Based on techniques developed in this WP, the practical implementation of a simple processor core is anticipated, which will present a proof of i-RISC concept viability. The current subtasks are presented in Table 1-1 and will evolve over time.

As illustrated in Gantt diagram presented in Table 1-1, during the first year we focused mainly on tasks 4.1, 4.2 and 4.3. Our activities include performance evaluation of Gallager B decoding algorithm and Taylor-Kuznetsov memory architecture based on structural LDPC codes, under independent and data-dependent logic gate failures. Also, analytical expression for bit error rate of one-step majority logic decoders built from unreliable logic gates is derived, which enable faster analysis of regular LDPC codes compared to Monte Carlo simulation. In addition, a new class of bit flipping algorithms operating at perfect hardware is proposed. These solutions belong to a set of low-complexity decoders, which makes them good candidates for fault-tolerant memories. Research conducted in an area of error-floors of iterative decoders represents a good background for faulty iterative decoder analysis.

Table 1-1: Gantt diagram of the WP 4.



2. Executive Summary

In this chapter we present a short summary of the activities carried out by the WP4, during the first year of i-RISC project and corresponding deliverables. We briefly discuss the most important technical contributions of our work with highlights on their relevance to the overall i-RISC project strategy.

The results of Subtask 4.1.a represent empirical evaluation of the performance of LDPC codes constructed from circular matrices (quasi-cyclic LDPC codes) decoded using the Gallager B decoder built from unreliable components. We assumed an independent transient fault model in which errors occur at Tanner graph level of implementation. Although faulty Gallager B decoder has been previously analyzed in the literature, using density evolution and EXIT function, the guidelines for designing a good decoder are not known. We examined the influence of different code parameters, decoder structures and fault model parameters to overall system performance in order to gain insight in relative importance of failures in different logic gates as their relation with parameters such as code length and number of decoding iterations. These results are of importance to designing optimal TK-based memory architectures, which is our next research topic.

The conducted Monte Carlo simulations have shown that the faulty Gallager B decoder is more sensitive to errors that occur in variable nodes than to errors in check nodes. The main reason for this behavior is related to variable node ability to compensate the parity check failures. The majority voting conducted in variable nodes can correct a fraction of parity check failures. However, if a variable node output is erroneous the correction ability of a decoder is decreased. The results indicate that the decoder performance can be significantly improved by better protection of variable nodes (e.g. by making the majority voting gates more reliable). Also, it is concluded that when the errors inserted into decoder are frequent, a codeword length may have negative impact to overall performance. Thus, increasing codeword length for the same code rate is not always beneficial. It is also interesting that if the check node faults are dominant, then the performance cannot be improved significantly by increasing the number of iterations. The codes with lower rates can correct more channel errors and therefore their performance is less degraded by decoder failures.

Independent transient failure model is only a rough approximation of the actual logic gate failures caused by increased noise sensitivity or/and timing constrains in new nano-scale semiconductor technologies. Although, actual failure rates are highly dependent on a digital circuit manufacturing technology, a digital circuit can be affected by multiple errors that exhibit correlation. The error modeling is done within Subtask 4.1.b. We present a novel approach to a faulty logic gate modeling which assumes output logic gate error dependence on several consecutive input values. To model such errors we use Markov chains. We investigated the influence of data-dependent failures on performance of majority voting and multiple input XOR gates, which are basic components of TK memory architecture. We determined the average performance of the TK scheme and identified cases in which our model can be simplified to independent transient failure model.

By enriching our failure model with the failure rate data obtained by measurements, the actual performance of memory architectures based on majority voting (TK memories, bit flipping based memories) can be evaluated. Furthermore, in the next phases of the project we will direct our research towards incorporating such more realistic failure conditions into our design. Competition of this task will be result of collaboration with UCC and UPT teams, in which research scope is faulty gate analysis.

One-step majority logic decoders are iterative decoders in which decoding process is terminating after only one cycle (iteration) and the bit estimates are obtained by a majority vote on multiple parity check decisions. Due to its simplicity, one-step decoder is a good candidate for ensuring memory reliability. Also, in contrast to iterative decoders, its performance can be evaluated analytically.

Our work, listed as Subtask 4.1c, includes determination of the faulty one-step majority logic decoder performance, when the component failures are modeled by a Markov chain. Based on combinatorial interpretation of decoding process, exact closed-form BER expression is derived for any regular LDPC code with girth length at least 6. Proposed analytical method has enabled us to accurately evaluate the bit error rate – much faster than by Monte Carlo simulation. Different regular LDPC codes have been examined and pronounced BER dependence of codewords order of decoding is noticed. Our later research is directed to finding optimal mapping strategies that minimize negative effects of date-dependence.

Bit flipping algorithms are the fastest and least complex among iterative LDPC decoders. But, their error performance is typically inferior compared to hard-decoding message passing algorithms such as the Gallager A/B algorithm. In Subtask 4.2a we present a novel class of bit flipping algorithms, called two-bit bit flipping (TBF) algorithms, which significantly reduce bit error rate, compared to standard bit flipping method, with only slightly more complex architecture. The proposed algorithms employ one additional bit at a variable node to represent its “strength”. The introduction of this additional bit allows an increase in the guaranteed error correction capability. An additional bit is also employed at a check node to capture information which is beneficial to decoding. Amazingly, efficient combining of additional information enable TBF algorithms to outperform, in some cases, message passing algorithms such as Gallager A/B or min-sum.

By constructing the TBF trapping set profile we identified the decoder failures and analytically predicted guaranteed error correction with high probability. Furthermore, we found different TBF algorithms that are capable of correcting different error patterns and introduced collective error correction. The concept and explicit construction of trapping set profiles allows rigorous selections of multiple algorithms which, operating in parallel, can collectively correct a fixed number of errors with high probability.

Although our analysis at this stage is restricted to decoders built entirely from reliable components, these findings are also applicable in the faulty decoder case. We expect that when comparing two equally complex decoders, the one with lower error level will also work better on noisy hardware. Our future work will be directed to proving aforementioned hypothesis and comparing performance of faulty TBF algorithms with noisy versions of other iterative decoders.

In Subtask 4.3a we analyzed iterative decoder failures and approaches for increasing LDPC codes performance in error-floor region. Based on the progressive constructing of Tanner graphs by avoiding trapping sets, we construct a LDPC code with same row and column weights, but superior error-floor performance compared to well known Tanner code. Again, we are investigating correspondence between code performance obtained under assumption of reliable computing operations and results derived on faulty hardware. We predict that better codes will also be superior under unreliable computing and we will next try to prove or disprove presented hypothesis.

In Subtask 4.3b we proposed a novel class of fast convergence iterative decoders called decimation-enhanced finite alphabet iterative decoders. By deleting a number of nodes from code computation

tree we expedite the decoding process while maintaining the same error performance. Furthermore, we showed that provable guaranteed error-correction can be achieved in a finite and small number of iterations, which represent a first result on guaranteed error-correction of a message-passing decoder other than Gallager-B decoder, and will hopefully lead to superior TK-memories.

As previously discussed, our main interest during the first year of the project was on error modeling and analysis of the TK-based memory architectures. Also, we investigated a novel approaches for designing low complexity decoders and codes with good performance in error-floor region. Next, we compared different message passing and bit flipping algorithms in order to identify decoders that are the most resistant to hardware faults, described by independent and correlated error models. At the end of the project we will be able to identify best decoder classes for reliable memory architectures.

3. Taylor-Kuznetsov memory architectures based on structured LDPC codes

Abstract: This technical contribution is dedicated to analysis of memory architectures constructed based on TK principle. We first evaluated the performance of Gallager B algorithm, used for decoding quasi-cyclic low-density parity-check (QC-LDPC) codes, under unreliable message computation. Using Monte Carlo simulation we investigated effects of different code parameters to coding system performance, under binary symmetric communication channel and independent transient faults model. We next presented a method for unreliable logic gates analysis in the presence of correlated, data-dependent gate failures, described by Markov chain model. The method is used for simulation analysis of Taylor-Kuznetsov memory architectures constructed from 2-input logic gates. Also, a theoretical analysis of one-step majority logic decodable codes in the presence of correlated gate failures is conducted. The closed-form expression for average bit error rate of any regular LDPC code with girth at least six has been derived.

3.1 Summary of the Related Work

According to new design paradigm for VLSI (Very Large Scale Integration) technologies, fully reliable operations are not guaranteed [Ghosh10]. New nano-scale technologies are more sensitive to noise, which appears as a consequence of radiation or electromagnetic interference. Thus, analysis of different decoding algorithms under unreliable hardware is meaningful. A hardware component is assumed to be unreliable if it is subject to so-called transient faults, i.e. faults that manifest themselves at particular time instants but do not necessarily persist for later times [Hadjicostis05]. These faults have probabilistic behavior and can be described statistically through erroneous component output probability.

Recently, different noisy LDPC decoders were analyzed by using simulation, density evolution or EXIT chart tools. The performance of LDPC codes under faulty Gallager-A and belief propagation decoding were determined in [Varshney11], using density evolution method. Similar analysis using EXIT function is provided in [Leduc-Primeau12], for Gallager B algorithm. Also, probabilistic analysis of Gallager B decoding algorithm was presented in [Yazdi13]. More general finite-alphabet decoders were investigated in [Huang13], while noisy min-sum decoder realization was considered in [Ngassa13].

The fault-tolerant memories were examined by Taylor [Taylor68-a], [Taylor68-b] who proposed use of LDPC codes as restoration organs in faulty memories. His work was continued and refined by Kuznetsov [Kuznetsov73]. Recently, as understanding of LDPC codes increases, research in this direction continues ([Vasic07], [Chilappagari07], [Chilappagari08], [Ivkovic06]).

One-step majority decoders were analyzed in [Chilappagari06] and [Radhakrishnan07] where authors provided a combinatorial characterization of the error in the output of the decoder in the presence of independent hardware failures.

3.2 Performance Analysis of Faulty Gallager B Decoding of QC-LDPC Codes

3.2.1 Description of Faulty Gallager-B decoder

Decoding of LDPC codes is usually described on code's Tanner graph. The Tanner graph is bipartite graph composed of two sets of nodes – variable (bit) nodes and check nodes. Nodes, from a different

set, connected to a single node, are referred to as its neighbors. The degree of a node is the number of his neighbors. In a (d_v,d_c) regular LDPC code, each variable node has degree d_v and every check node degree is d_c.

The Gallager B algorithm represents iterative decoding procedure operating in a binary field. During the every decoding iteration, binary messages are sent along the edges of Tanner graph. Let $E(x)$ represent a set of edges incident on a node x (x can be either variable or check node). Let $m_i(e)$ and $m'_i(e)$ denote the messages sent on edge e from variable node to check node and check node to variable node at iteration i, respectively. If we denote the initial value of a bit at variable node v as $r(v)$, the Gallager B algorithm can be summarized as follows [Shokrollahi02].

Initialization (i=1): For each variable node v, and each set $E(v)$, messages sent to check nodes are computed as follows

$$m_1(e) = r(v). \tag{3-1}$$

Step (*i*) (*check-node update*): For each parity check node c and each set $E(c)$, update rule for i-th iteration, $i > 1$, is defined as follows

$$m'_i(e) = \left(\sum_{e\prime \in E(c)/e} m_i(e') \right) mod 2. \tag{3-2}$$

Step (*ii*) (*variable-node update*): For each variable node v and each set $E(v)$, update rule for i-th iteration, $i > 1$, is defined as follows

$$m_i(e) = \begin{cases} 1, & if \sum_{e\prime \in E(v)/e} m_i(e') \geq b_i, \\ 0, & if\ d_v - 1 - \sum_{e\prime \in E(v)/e} m_i(e') \leq b_i, \\ r(v), & otherwise, \end{cases} \tag{3-3}$$

where b_i represent threshold dependent on iteration i. In our analysis we considered constant threshold value $b_i = \lceil d_v/2 \rceil$.

Step (*iv*) (*decision*): After predefined number of iterations the final decision of transmitted bit $\hat{v}$ is made on the basis of majority of its estimates $m_i(e), e \forall\ E(v)$.

We studied the performance of a faulty Gallager B decoder in the presence of transient faults. In addition to noise that exists in communication channel, errors are inserted by the LDPC decoder itself. We assume independent transient faults model in which errors occur at the output of variable and check node update functions. In other words, every edge in Tanner graph behaves as binary symmetric channel (BSC) with some crossover probability. The probability that message originating from variable node is incorrect is denoted as p, while crossover probability in BSC that corresponds to check node message transition is equal to q. Assigning different crossover probabilities enable us to determine the influence of faults in different nodes to overall decoder performance.

3.2.2 Numerical Results

In this section we present performance analysis of faulty Gallager B decoder, described in the previous section. The two QC-LDPC codes have been examined and their performance are compared

for several implementations of faulty Gallager B decoders. All numerical results presented in this section are obtained by Monte Carlo simulations.

The sequence of all-zero codewords is transmitted through BSC with a predefined crossover probability and then decoded by a faulty iterative decoder. As described earlier, messages that are passed between nodes can be faulty. The message $m_i(e)$ passes through the noise channel with error probability p, thus, a bit estimate can be erroneous as a consequence of a majority of unsatisfied parity checks or the faults in variable node implementation or both. Similarly, due to BSC crossover probability q, message $m'_i(e)$ may incorrectly inform variable node if the parity check equation is satisfied or not.

First, we evaluated the performance of Tanner code (with n=155, d_v=3 and d_c=5) decoded by a faulty Gallager B decoder. The code frame error rate (FER) performance are given as a function of communication channel crossover probability. FER curves for several values of decoder failures probabilities p and q, when 5 decoding iterations are performed, are presented in Fig. 3-1. It can be observed that decoder failures greatly degrade frame error rate, but failures in variable and check nodes have different influence on the code performance. The simulation has shown that the decoder is more sensitive to errors that occur in variable nodes. The mean reason for this behavior is related to variable node ability to compensate the parity check failures. The majority voting conducted in variable nodes can correct a fraction of parity check failures. However, if a variable node output is erroneous correction ability of a decoder is decreased. Finally, the presented results indicate that the decoder performance can be significantly improved by better protection of variable nodes (e.g. by making the majority voting gates that perform the operation in (3-3) more reliable).

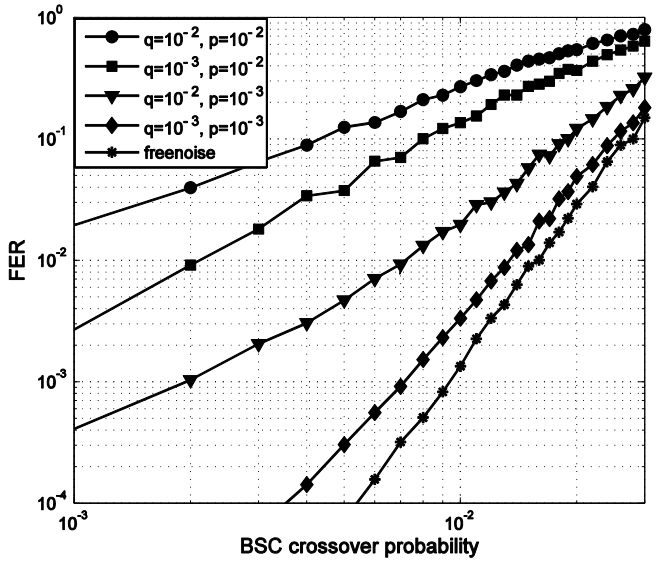


Figure 3-1: Performance of Tanner code (155,64) decoded by a faulty Gallager B decoder, five iterations.

We also evaluated performance of two QC-LDPC codes with code lengths n_1=155 and n_2=305, with the same parameters d_v=3 and d_c=5. Performance comparison is illustrated in Fig. 3-2. Although the code with longer codewords has better correcting capabilities, it is also more prone to processing errors. The simulation has shown, that when the errors inserted into decoder are frequent (p=10^{-2} or q=10^{-2}), longer code length may have negative impact to overall performance. However, the longer code achieves lower FER when hardware failures are rare (p=10^{-4}).

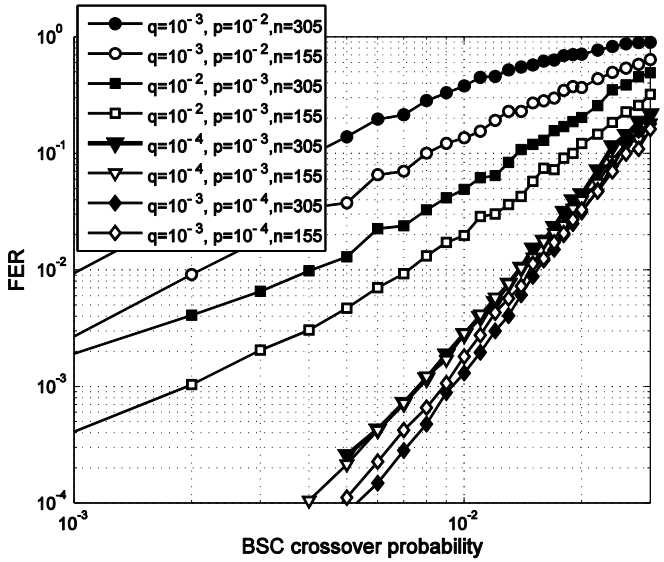


Figure 3-2: Performance comparison of two QC-LDPC codes with different lengths (n=155 and n=305) with d_c=5, d_v=3, decoded by a faulty Gallager B decoder in five iterations.

The performance of the Gallager B algorithm depends on the number of iterations [MacKay99], thus assessing the effect of number of iterations of faulty decoder is meaningful. The performance of a faulty decoder, when a different numbers of decoding iterations are used, are presented in Fig. 3-3. It is obvious that increasing the number of decoding iteration leads to lower error rates. However, it can be noticed that the improvement depends on the structure of the errors that exist in decoder.

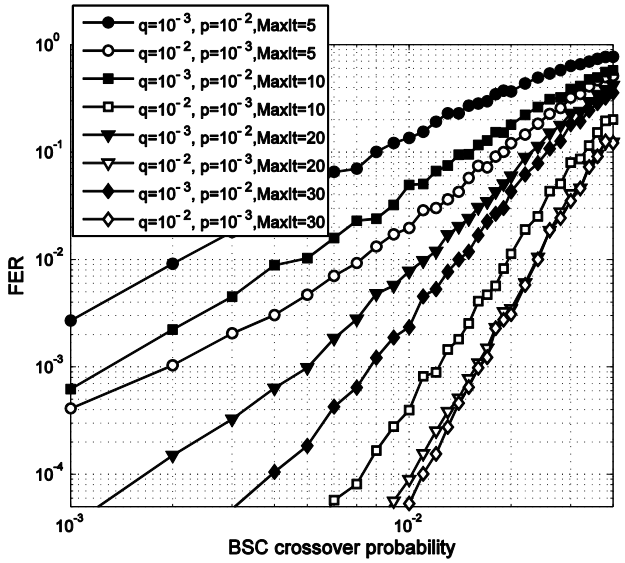


Figure 3-3: Influence of number of decoding iteration to faulty Gallager B decoder performance, Tanner code (155,64).

If the variable node faults are dominant (p=10^{-2}, q=10^{-3}), the performance can be improved significantly by increasing the number of iterations. In contrast, the FER levels are much lower if the check nodes faults have dominant effect, and cannot be improved significantly by increasing the number of iterations. For example, when the Tanner code is decoded by a faulty Gallager B decoder with parameters p=10^{-3} and q=10^{-2} only 20 decoding cycles is sufficient and the error rate does not further improve.

Finally, we investigated the influence of code rate on decoder performance. We compare the error rates of two QC-LDPC codes with the same length (n=155) and check node degree (d_c=5), but different variable node degrees (d_v=3 or d_v=4). The obtained results are presented in Fig. 3-4. The code with higher variable nodes degree (lower code rate) can correct more errors that appear in communication channel, but the decoder is also more complex and more prone to errors. It is interesting to notice that the performance of code with lower code rate is less degraded by decoder failures.

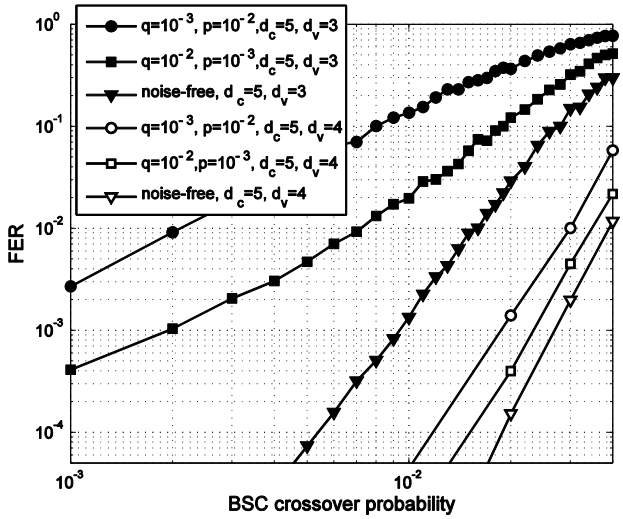


Figure 3-4: Performance comparison of two QC-LDPC codes of codeword length n=155 with d_c=5 and different column weight (d_v=3 and d_v=4), faulty decoding in five iterations.

3.3 Performance Analysis of Taylor-Kuznetsov fault-tolerant memories under correlated gate failures

3.3.1 The Correlated Failure Model

Taylor-Kuznetsov model assumes that faults of memory elements and logic gates are transient, i.e. faults occur at a particular time instants but do not necessary persist for later times. It is also assumes that gates fail independently of each other and that faults are not permanent, i.e. faulty gate may produce a correct output at some time instant. Such failure mechanism is referred to as von Neumann type of errors. In this paper we consider a more general failure model in which failures of a given logic gate are data-dependent and correlated in time.

Our model can be used for analyzing physical phenomena such as noise influence to chips with sub-threshold voltages. We formed a model of faulty gates through its first order statistics (probability of erroneous circuit output) as a function of logic gates inputs or outputs. For that purpose we used Markov chains.

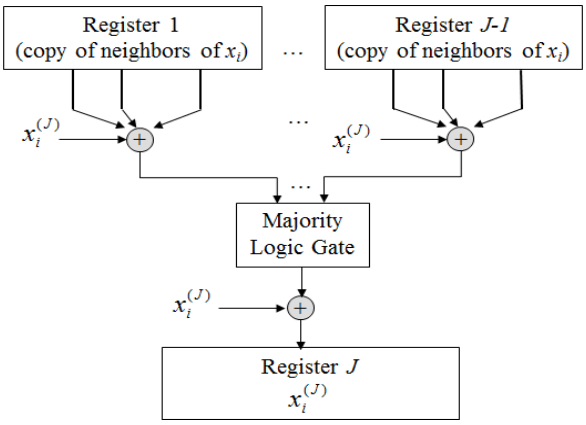


Figure 3-5: Block diagram of the Taylor-Kuznetsov memory architecture.

As it can be seen in Fig. 3-5, the TK scheme [Taylor68-a] is composed of (*J*-1)-input majority logic gates, *K*-input XOR gates and 2-input XOR gates. Because all multi-input gates in hardware are implemented as circuits composed of 2-input Boolean logic gates, our analysis is done at the 2-input Boolean function level. The correct binary output value of a given Boolean function, $O_c(k)$ at discrete time *k*, (*k*>0), depends on $I_1(k)$ and $I_2(k)$, the logic gate binary input values. This is illustrated in Fig. 3-6. The errors are inserted into logic gate by performing XOR operation between correct gate output sequence $\{O_c(k)\}_{k>0}$, and the error sequence $\{e(k)\}_{k>0}$, producing the actual output sequence $\{O_e(k)\}_{k>0}$, (Fig. 3-6). The error sequence $\{e(k)\}_{k>0}$, represents the binary time series which describes the statistics of errors. If the *k*-th value of error pattern is '1', i.e. $e(k)=1$ ($k>0$), the output of Boolean logic gate at time *k* will be faulty, i.e., the *k*-th actual output value will not correspond to correct one ($O_e(k)\neq O_c(k)$). The error sequence is modeled as a finite Markov chain. The error pattern statistics depends on gate inputs and outputs, and in principle there are two ways to define such dependence. In the first approach the $e(k)$ at discrete time *k* taking some value depends on the current gate output $O_c(k)$ as well as *M, M*>0, previous gate outputs $O_c(k\text{-}1)$, … $O_c(k\text{-}M)$. In the second approach, the probability of error pattern depends on gate inputs $I_1(k)$, $I_2(k)$, $I_1(k\text{-}1)$, $I_2(k\text{-}1)$... $I_1(k\text{-}M)$, $I_2(k\text{-}M)$). In this section, our focus was on output dependence model because its complexity is half of that of the second model, while still capturing the essential characteristics of data dependence.

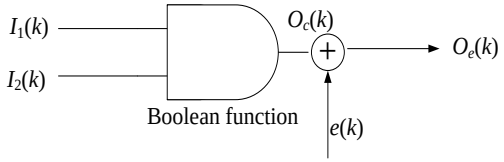


Figure 3-6: Error insertion into a 2-input Boolean function.

Let *S* be a Markov source generating the error sequence composed of 2^{M+1} states s_i, $1\leq i\leq 2^{M+1}$, i.e. $S=\{s_1, s_2, \ldots, s_{2^{M+1}}\}$ Every state corresponds to one possible logic gate output sequence of length *M*+1 and captures the different data-dependent failures, expressed through different error probabilities $p(s_i)=\Pr\{e(k)=1|s_i\}$, $1\leq i\leq 2^{M+1}$, $k>0$, where Pr{.} denotes probability. According to current binary output value $O_c(k)$, $k>0$, from each state s_i only transition to two next states s_j and s_m are possible, which happens with probabilities p_{ij} and p_{im}, respectively. The transition probabilities p_{ij}, depend on statistic of sequence $\{O_c(k)\}_{k>0}$, and must satisfy $p_{ij}+p_{im}=1$, $1\leq a\leq 2^{M+1}$, $a\in\{i,j,m\}$. Although large value of *M* can give us possibility to describe data-dependence in more details, complexity of model increases exponentially. Also, long correlation is not expected in faulty logic gates. Thus, we

use the simplest model with M=1, which can adequately illustrate data-dependent failures. We next analyze AND, OR and XOR faulty logic gates.

At the output of AND logic gate correct value '1' appears only if both inputs are equal to '1'. Thus, if due to increased noise level one input changes its value, the gate output will be faulty. We can conclude that gate output '1' is more prone to errors. Consequently, the error probability in the state s_4='11' is the highest and in the s_1='00' state is the lowest. The last conclusion can be formulated as follows

$$p_{AND} = \Pr\{e = 1|s_i\} = A_i \Pr\{e = 1|s_4\}, \tag{3-4}$$

where $p_{AND}(s_i)$ denotes the error probability at the output of AND logic gate in state s_i, $1 \le i \le 4$, and A_i represent the scaling coefficients, dependent of state s_i. From the discussion presented above, it is clear that must hold

$$A_1 \le A_0 \le A_4, A_0 \in \{A_2, A_3\}. \tag{3-5}$$

Similar analysis can be performed for the OR logic gate. The correct output value '0' can be changed to incorrect value '1' even if only one input is faulty. Thus, in this case the state '00' is the most sensitive and we can write the following expression

$$p_{OR} = \Pr\{e = 1|s_i\} = B_i \Pr\{e = 1|s_0\}, \tag{3-6}$$

where $p_{OR}(s_i)$ denotes the error probability at the output of OR logic gate in state s_i, $1 \le i \le 4$, and B_i represent the scaling coefficients, dependent of state s_i. It can be noted that following condition must be satisfied

$$B_4 \le B_0 \le B_4, B_0 \in \{B_2, B_3\}. \tag{3-7}$$

Every change of input values of XOR logic gate will produce an error. Thus, probabilities of output error will be the same regardless of current state, $p_{XOR}(s_i)=p_{XOR}$, $1 \le i \le 4$, i.e. XOR error pattern can be modeled as BSC.

As previously described, using AND, OR and XOR logic gates every multiple-input logic gate in TK scheme can be implemented. For example, faulty 3-input majority logic gate can be presented as a digital circuit composed of three faulty AND gates and two faulty OR gates.

The results of majority logic gates analysis are graphically presented in Fig. 3-7 and Fig. 3-8, for several values of A_i and B_i coefficients and inputs statistics. The input statistics are described by probability that input values I_i (i=1,2,3) are equal to '1', denoted as P_1. The failure model coefficients are given in normalized form as follows

$$B_1 = A_4 = 1, B_2 = B_3 = A_2 = A_3 = \frac{1}{p}, B_4 = A_1 = \frac{1}{p^2}. \tag{3-8}$$

In this way all coefficients are described by a single parameter (p). It should be noted that this coefficient relations are purely theoretical and they have not been validated by real measurements.

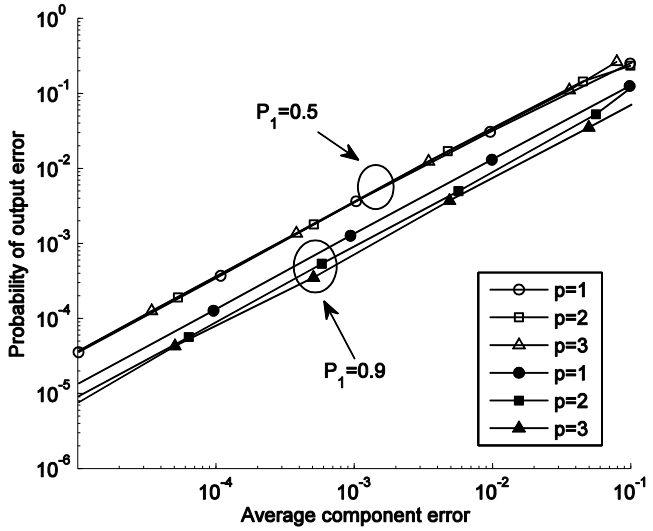


Figure 3-7: Probability of error in 3-input majority logic gate.

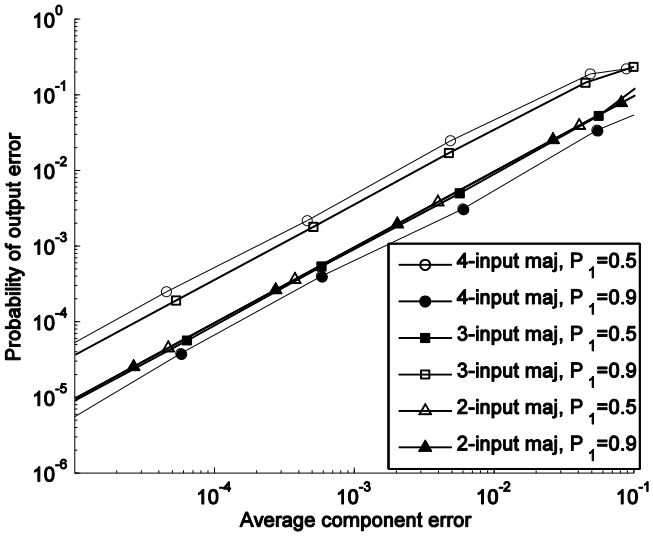


Figure 3-8: Comparison of majority logic gates with different number of inputs for p=2.

The output error probability of 3-input majority logic gate dependence of average component failures is presented in Fig 4-3, for several values of parameter p(=1,2,3) and two input probabilities P_1=0.5 and P_1=0.9. It can be noticed that input statistics have the most influence on logic gate performance. A majority logic gate output is equal to '1', if half or more inputs are equal to '1'. Thus, when ones and zeros appear at the gate inputs with equal probabilities (P_1=0.5) more gate output values will be faulty, compared to case when almost all inputs are '1' (P_1=0.9). When P_1=0.5, parameter p, which describes presented Markov model, does not have any impact on logic gate performance. So, for that case the presented model is excessive and can be replaced by uncorrelated error model. When P_1=0.9 differences caused by error correlation exist.

Furthermore, as the output error probability is a linear function of average component error, a faulty majority logic gate can be modeled as correct one at which output the error pattern is inserted. Thus, when P_1=0.5, a simpler model can be used, which is characterized only by output error probability, as it is presented in Fig. 3-9.

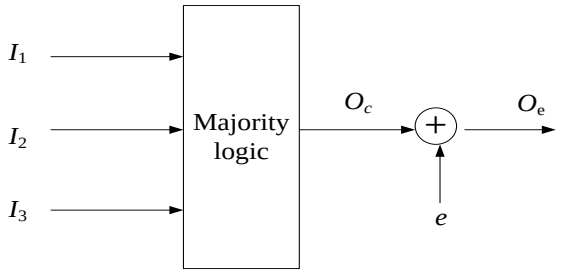


Figure 3-9: Equivalent scheme of faulty 3-input XOR logic gate.

It should be noted that simplification, presented in Fig. 3-9, is accurate only if autocorrelation function of error pattern is unchanged. But, in this case, as simulation analysis has shown, presented Markov chain produced the output error pattern which greatly resembles the uncorrelated one. Thus, it is sufficient to determine first order statistics of the error pattern.

The performance comparison of majority logic gates with different number of inputs is presented in Fig. 4-4, when p=2. The number of inputs of majority logic gate used in TK scheme depends of column weight of used LDPC code parity check matrix. Thus, the comparison of different majority logic gates can give us an insight what LDPC code will produce less errors in the decoding phase. Also, code correcting capabilities have a significant influence in choosing the right code for TK scheme. It can be noted that 2-input majority logic gate (which is actually a simple OR logic gate) has the lowest output error probability when P_1=0.5. But, when P_1=0.9, the simulation has shown that the gate with largest number of inputs (4-input gate) outperforms other logic gates. The gates with more inputs are less sensitive to errors when input value '1' is more frequent then value '0'.

We can conclude that input values statistics have a key role in majority logic gate analysis. If we can determine the probabilities of input values we can easily generate a simpler model of faulty majority logic gate and choose the LDPC code, which will be more resistant to hardware failures.

The faulty multi-input XOR gates are also an integral part of the TK scheme and, as already mentioned, can be represented as 2-input faulty XOR gates. The number of inputs of XOR logic gate is defined by the row weight of chosen LDPC code parity check matrix. Thus, it is interesting to compare XOR gates with different number of inputs.

Performance of XOR gates with 3, 4 and 5 inputs are presented in Fig 3-10. It can be noted that increasing the number of inputs causes higher output error probability. In multi-input XOR gate every odd number of 2-input XOR gate failures will produce output errors. In XOR logic gates with more inputs more 2-input XOR gate failure combination can generate an output error.

Because errors in 2-input XOR gates are uncorrelated, the number of inputs is only parameter that affects multi-input XOR gate performance. The input values statistics do not have any impact on output error probability in our XOR gate failure model.

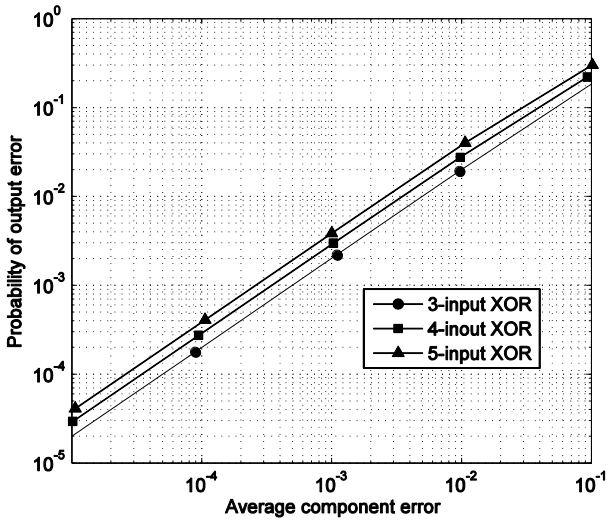


Figure 3-10: Comparison of XOR logic gates with different number of inputs (output dependence model).

3.3.2 Numerical Results

In this section we present the simulation results for TK memory architecture with codeword length equal to 15 bits. We used LDPC code constructed by Euclidean geometry principle, EG(15,7). The parity check matrix of this code has four 1's in every row and column. The length of the shortest cycle of the Tanner graph representation is equal to 6.

We assumed that memory failures are independent and they can happen with probability P_m. The failures of correcting circuit are modeled by using Markov model described in the previous section. Thus, the probability that output of Boolean function is incorrect in the worst state P_b, $P_b=p_{AND}(s_4)=p_{OR}(s_1)=p_{XOR}$, and value of parameter p from Eq. 3-8 completely defines the failure model.

We assumed that the memory contents pass through a BSC after a time period T and then are updated by the message passing Gallager B decoder. Initially, all-zero code word is stored in memory registers. The message passing can be run for any number of iterations. It is also assumed that the time for update is smaller compared to T and that memory contents do not change while the update is in progress.

The bit error rate (BER) curves for described memory architecture when $P_m=10^{-3}$ and $p=2$, are presented in Fig. 3-11, for several values of P_b. The update process is terminated after four iteration of Gallager B algorithm. It can be noted that when correcting circuits faults are of the same order of magnitude as memory elements faults, the reliable memory cannot be achieved. If logic gate error probability has lower values ($P_b=10^{-4}$, $P_b=10^{-5}$) BER does not increase rapidly and stays below memory error probability for several time steps T.

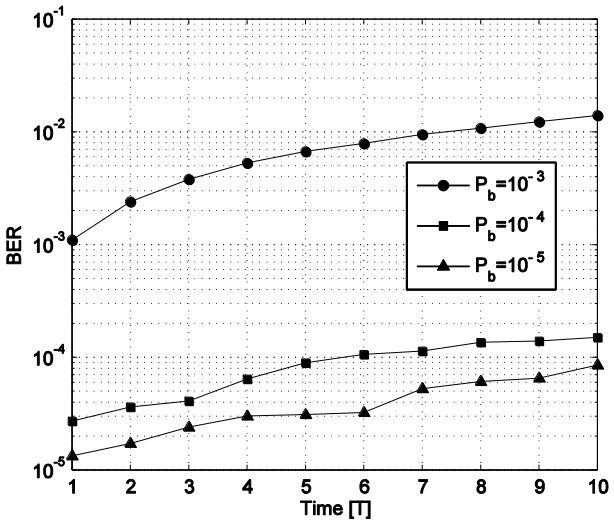


Figure 3-11: Performance of TK scheme with EG(15,7) LDPC code in a correlated error model (p=2).

3.4 Analysis of One-step Majority Logic Decoding under Correlated Data-Dependent Gate Failures

3.4.1 Description of One-step Majority Logic Decoders and Failure Model

Let C be the (γ,ρ)-regular LDPC code of length n, with parity check matrix H which has exactly γ 1's in every column and ρ 1's in every row. A codeword $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is stored in a memory, and when read from the memory each bit x_v is flipped by probability α and observed as r_v. We refer to r_v as value of the variable node v.

Let E_x represent a set of edges incident on a node x in a Tanner graph (x can be either variable or check node), then the one-step majority decoding may be summarized as follows.

1. Each variable node v sends r_v along the edge in E_v.
2. Each check node c sends m_e along each edge in E_c where

$$m_e = \sum_{e' \in E_c/e} m_{e'} \mod 2. \tag{3-9}$$

3. At each v an estimate of the value of $\widehat{r_v}$ is made

$$\widehat{r_v} = \begin{cases} 1, if\ |\{e' \in E_v : m_{e'} = 1\}| > \lfloor \gamma/2 \rfloor \\ 0, if\ |\{e' \in E_v : m_{e'} = 0\}| > \lfloor \gamma/2 \rfloor \\ r(v), \quad otherwise \end{cases}. \tag{3-10}$$

Note that, similarly as in [Chilappagari06-a], we only analyze the message passing version of one-step majority logic decoder. This decoder needs needs γ of (ρ – 1)-input XOR gates at each check node and a γ-input majority logic gate at each variable node. Each check node makes an estimate of the value of a variable node based on the values of other variable nodes. The value of the variable node itself is not used in its estimation. The final decision is made on the basis of majority of the estimates, resulting in probability of error of an estimated bit greater than or equal to the probability of failure of the majority logic gate. Since the error probability of the majority logic gate lower bounds the bit error rate (BER) performance, majority logic gates must be made highly reliable. Otherwise, the probability of error is determined by this final gate and not the error control scheme. Thus, it is

reasonable to make an assumption that majority logic gates are perfect and that only XOR gates are faulty. We analyze a system in which different codewords are stored in an unreliable memory. While stored in the register, each bit may be flipped, independently of other bits, with probability α. At every cycle a different codeword is read from the memory and decoded by an unreliable one-step majority decoder. Equivalently, we may assume that the sequence of codewords $\{\mathbf{x}^{(k)}\}_{k\geq 0}$ is transmitted through the Binary Symmetric Channel (BSC) with crossover probability α and then successively decoded by a single one-step majority decoder built from perfect majority logic gates and faulty XOR gates.

In contrast to the state-of-the-art modeling of faulty gates that considers only the failure dependence on the current input values, our model captures more accurately the data and time dependence of the failures. Namely, we assume that $e^{(k)}$, the error at time k is affected by the current and M-1 prior consecutive gate input vectors, i.e., its probability depends on the vector sequence $\{\mathbf{y}^{(j)}\}_{j\in[k-(m-1),k]}$,where M is a positive integer and $\mathbf{y}^{(k)} = (y_1, y_1, \dots, y_{\rho-1})$ represents vector of gate input values at time k. Denote this probability by $\Pr\{e|\mathbf{s}^{(k)}\}$, where the gate state $\mathbf{s}^{(k)}$ at time k is defined as $\mathbf{s}^{(k)} = \{\mathbf{y}^{(j)}\}_{j\in[k-(m-1),k]}$. As previously stated, in our one-step majority logic decoder only XOR gates are unreliable. The number of states grows exponentially with M and ρ, i.e., for an (ρ-1)-input XOR gate used in our decoder there are $2^{M(\rho-1)}$ states.

The inputs of a (perfect) majority logic gate are the outputs of XOR gates in the neighboring check nodes. Thus, at time k these gates can be associated with a state array $\sigma^{(k)} = \left(s_1^{(k)}, s_2^{(k)}, \dots, s_\gamma^{(k)}\right)$, whose elements represent states of particular XOR gates. Based on $\sigma^{(k)}$, an error probability vector can be formed as $\epsilon^{(k)} = \left(\epsilon_1^{(k)}, \epsilon_2^{(k)}, \dots, \epsilon_\gamma^{(k)}\right)$, $\epsilon_m^{(k)} = \Pr\{e = 1|\mathbf{s}^{(k)}\}, 1 \leq m \leq \gamma$. The values of error probability vector can be obtained by measurements or by simulation of selected semiconductor technology. Thus, in our analysis we assumed that these values are known.

3.4.2 Performance Analysis

A composition of a positive integer i, is an integer vector $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_\gamma)$ such that $\sum_{j=1}^{\gamma} \lambda_j = i$. The integers $\lambda_j \geq 0$ are called parts or summands, and γ is called the composition length. Note that we slightly modified the standard definition of the composition by allowing the parts to be equal to zero. Consequently, all compositions have the same length, which simplifies representation of error patterns.

Let $[l]$ denote the set of first l positive integers, i.e., $[l] = \{1,2, \dots, l\}$. A restricted composition in which no part size exceeds l is an $[l]$-composition. The number of the compositions of an integer i whose part sizes do not exceed a fixed integer l is denoted by $T_{i,l}$ and can be calculated based on the methods given in [Malandro13]. Alternatively, an $[l]$-composition can be rewritten using its frequency representation $(1^{f_1}, 2^{f_2}, \dots l^{f_l})$, where f_j denotes the number of parts equal to j,$1 \leq j \leq l$. For example, (1,2,1,1) is an $[2]$-composition of 5 of length 4, with frequency representation f_1 = 3, f_2 = 1.

Let $\mathbf{q}_l$ be a vector corresponding to one lexicographically ordered u-subset of the set $[l]$ and let the vector $\mathbf{q}_r$ contain the remaining elements of $[l]$. We create a vector $\mathbf{q}$ by juxtapositioning $\mathbf{q}_l$ and $\mathbf{q}_r$. We can arrange all possible vectors $\mathbf{q}$ into rows of an $\binom{l}{u}$ by l array $Q^{u,l}$. For example, if $l = 4$ and $u = 2$, the rows of $Q^{2,4}$ are (1,2,3,4), (1,3,2,4), (1,4,2,3), (2,3,1,4), (2,4,1,3) and (3,4,1,2). The array

$Q^{u,l}$ referred to as the error configuration matrix will be instrumental in book-keeping of data-dependent patterns.

In a Tanner graph of a code with girth at least six, the variable nodes connected to the neighboring checks, E_v, of a variable node *v*, are all distinct. Without loss of generality we can label the check nodes as $E_v = [\gamma]$, and the set of the $\gamma(\rho - 1)$ variable nodes by the elements of $[\gamma(\rho - 1)]$. Consider now a weight i $(i > 0)$ error pattern on these variable nodes, and consider the computation tree for a variable node *v*, presented in Fig 3-12. It is a tree with the root *v* and the leafs in $[\gamma(\rho - 1)]$, i out of which are erroneous. The distribution of the weight i error pattern on the leaf nodes of the computation tree can be represented by a restricted $[\rho - 1]$-composition of i with parts. Each part $\lambda_m, 1 \leq m \leq \gamma$ represents the number of erroneous nodes connected to the *m*-th check node (XOR gate). Since each check node in E_v is connected to $\rho - 1$ variables other than *v*, the part size cannot exceed $\rho - 1$. For example, if $\gamma = 3$, $E_v = \{1,2,3\}$ and $i = 3$, $\lambda = (2,0,1)$ corresponds to an error pattern in which the errors are at $\lambda_1 = 2$ variable nodes connected to the check $c = 1$ and $\lambda_3 = 1$ variable nodes connected to the check $c = 3$, while no variable connected to $c = 2$ is in error.

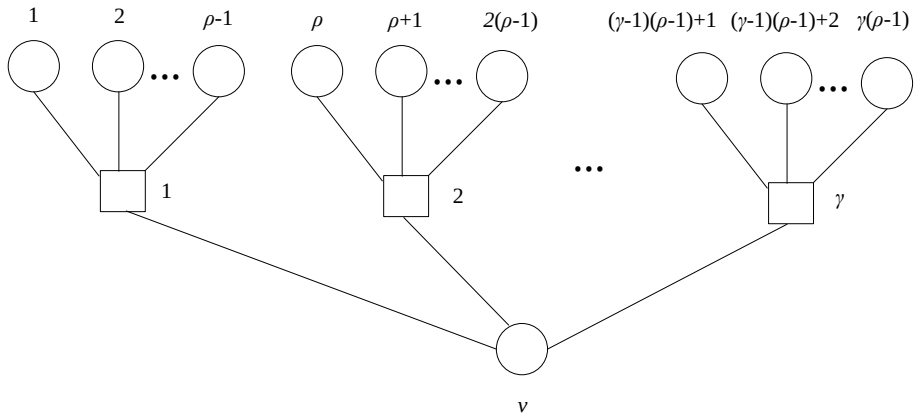


Figure 3-12: Part of Tanner graph used for decoding bit *v*.

In a decoder built entirely from reliable components, every odd number of erroneous neighboring nodes connected to the same XOR gate will result in sending an incorrect bit estimate to node *v*. The number of incorrect estimates sent to node *v* in the error pattern λ can be obtained as follows

$$\delta = \sum_{t=1}^{[\rho-1/2]} f_{2t-1}. \tag{3-11}$$

In a perfect decoder determining the value δ is sufficient to know if bit x_v is correctly decoded, assuming the error pattern λ. But, in the faulty decoder, due to XOR gate failures, some of the correct estimates can become incorrect, and vice versa, some incorrect estimates can be received as correct ones. Let ϵ be the error probability vector of unreliable XOR gates, used for decoding bit x_v. Denote as $\epsilon^0 = (\epsilon_1^o, \epsilon_2^o, \dots, \epsilon_\delta^o)$ a vector of error probabilities that correspond to all XOR gate which, if operate perfectly, would send incorrect bit estimates to node *v*. The remaining $\gamma - \delta$ elements of the error probability vector ϵ form a vector ϵ^e. Then we have the following lemma establishing the harmfulness of the error pattern corresponding to the composition λ.

Lemma 3.1: The probability that node *v* receives *p* incorrect bit estimates in an error configuration corresponding to λ is

$$P(p,\lambda,\epsilon)=\sum_{u=u_l}^{u_h} P_w\{u,\delta\}P_c\{p-u,\gamma-\delta\}, \tag{3-12}$$

where u_h=min(δ,p), u_l=max(0,$p-\gamma+\delta$),

$$P_w\{u,\delta\}=\sum_{t=1}^{\binom{\delta}{u}}\prod_{m=1}^{u}\left(1-\epsilon_{q_{t,m}}^{0}\right)\prod_{m=u+1}^{\delta}\epsilon_{q_{t,m}}^{0} \tag{3-13}$$

and

$$P_c\{p-u,\gamma-\delta\}=\sum_{t=1}^{\binom{\gamma-\delta}{p-u}}\prod_{m=1}^{p-u}\epsilon_{r_{t,m}}^{e}\prod_{m=p-u+1}^{\gamma-\delta}\left(1-\epsilon_{r_{t,m}}^{e}\right), \tag{3-14}$$

where $q_{t,m}$ and $r_{t,m}$ denote the elements in t-th row and m-th column of the error configuration matrices $Q^{u,p}$and $Q^{p-u,\gamma-\delta}$, respectively.

The following lemma gives the bit miscorrection probability under the fixed error probability vector, when the effect of different memory failure configurations is taken into account.

Lemma 3.2: If memory registers fail independently with probability α, then the probability that the codeword bit x_v of (γ,ρ)-regular LDPC code is incorrectly decoded by a faulty one-step majority logic decoder, whose gate fail according the error probability vector ϵ, is given by

$$P_v(\alpha,\epsilon)=\sum_{i=0}^{\gamma(\rho-1)}\alpha^i(1-\alpha)^{\gamma(\rho-1)-i}\left[\frac{(-1)^\gamma+1}{2}\alpha b_{i,\lfloor\gamma/2\rfloor}+\sum_{p=\lfloor\gamma/2\rfloor+1}^{\gamma}b_{i,p}\right], \tag{3-15}$$

where

$$b_{i,p}=\sum_{j=1}^{T_{i,\rho-1}}P\left(p,\lambda^{i,j},\epsilon\right)\prod_{m=1}^{\gamma}\binom{\rho-1}{\lambda_m^{i,j}}, \tag{3-16}$$

and $\lambda^{i,j}=(\lambda_1^{i,j},\lambda_2^{i,j},\dots,\lambda_\gamma^{i,j})$ is the j-th composition of integer i, where $1\le j\le T_{i,\rho-1}$.

Let$\left\{\mathbf{x}^{(k)}\right\}_{k\ge0}$ be a codeword sequence stored in the memory registers. Clearly, decoding error of $\mathbf{x}^{(k)}$ depends on $M-1$ codewords previously read from a memory. Let $\mathbf{y}_{m,v}=\left\{\mathbf{y}^{(j)}\right\}_{j\in[k-(m-1),k]}$ $1\le m\le\gamma$ $1\le v\le n$, be sequence of code bits that, if stored with no errors, will appear at inputs of m-th XOR gate connected to node v, in the time interval $[k-(m-1),k]$. Then, using the Lemmas 1 and 2 we formulate our main theorem which captures decoder performance under correlated data-dependent gate failures.

Theorem 3.1: The average bit error rate (BER) of (γ,ρ)-regular LDPC code, when codeword sequence $\left\{\mathbf{x}^{(j)}\right\}_{j\in[k-(m-1),k]}$ is read from a memory and decoded by an unreliable one-step majority logic decoder is

$$\overline{P}_e\left(\mathbf{x}^{(k)}|\mathbf{x}^{(k-1)}\dots\mathbf{x}^{(k-M+1)}\right)=\frac{1}{n}\sum_{v=1}^{n}\sum_{t=1}^{2^{(\rho-1)M\gamma}}P_v\left(\alpha,\epsilon^{(t)}\right)\prod_{m=1}^{\gamma}\alpha^{d_H\left(s_m^{(t)},\mathbf{y}_{m,v}\right)}(1-\alpha)^{M(\rho-1)-h\left(s_m^{(t)},\mathbf{y}_{m,v}\right)}, \tag{3-17}$$

where $d_H(a,b)$ denotes Hamming distance of binary vectors a and b.

For a special case of von Neumann errors, the probability $P_v\left(\alpha, \epsilon^{(t)}\right)$ is independent of state arrays and above expression reduces to equation (3-15). In addition, for the perfectly reliable hardware, the equation (3-12) simplifies to $P(p, \lambda, \epsilon) = 1$ only when $p = \delta$, and it is equal to zero otherwise.

The analysis presented in this section is general and can be used for analyzing decoders built in different nanoscale technologies. In this section we present numerical results for a special case of transient errors that are result of timing constrains - *timing errors*. Due to the sampling clock fluctuations or signal propagation delays, the output signal of a gate may be sampled or used in the next stage before it reaches a steady value, leading to an incorrect output. Such errors are dependent on gate history, i.e. data values processed by the gate in previous bit intervals. As the erroneous output of a logic gate will appear only if output changes its values it is usually sufficient to consider failure dependence on current and only one previous bit interval, i.e. $M = 2$. Thus, two subsets of input value states of faulty XOR gate can be identified. First subset is constituted by states in which gate output remains unchanged and zero failure rates correspond to this subset. If gate output changes its value failure is possible and states from other subset have non-zero failure rates. For simplicity, we assume that all failure rates from second subset are the same, denoted as ε.

The performance of the faulty decoder for two limiting cases are presented in Fig. 3.14. The best case, denoted as X_{00} corresponds to storage and consecutive decoding of two same codewords. The decoder will operate worst if two complementary codewords are stored, denoted as X_{01}. Performance of faulty decoders are upper bounded under decoding of X_{01} and lower bounded under X_{00}. When decoding X_{00} XOR gate failures are rare and have limited influence on decoder performance, which results in practically the same BER values for both ε values (ε=10^{-3},10^{-2}). Gap between bounds depends on column weight of matrix H. The upper bounds for different (γ,ρ) classes of LDPC codes are presented in Fig. 3.15, where negative influence of matrix H row weight can be noticed.

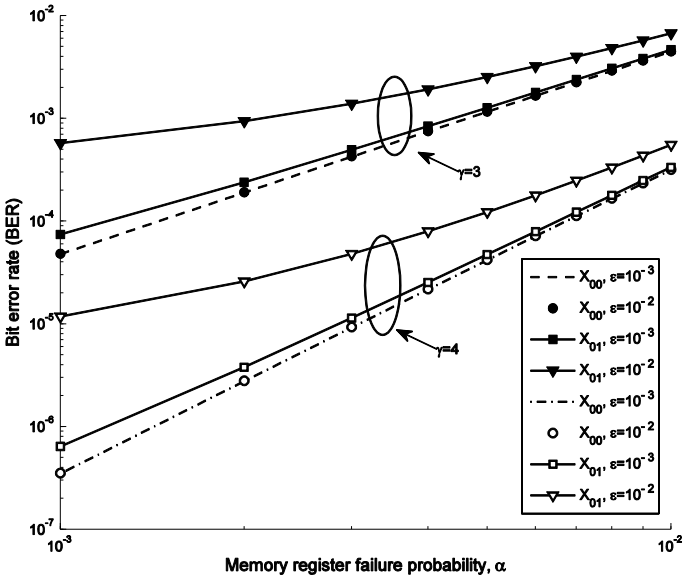


Figure 3-14: Performance of faulty decoder under correlated gate failure (ρ=5).

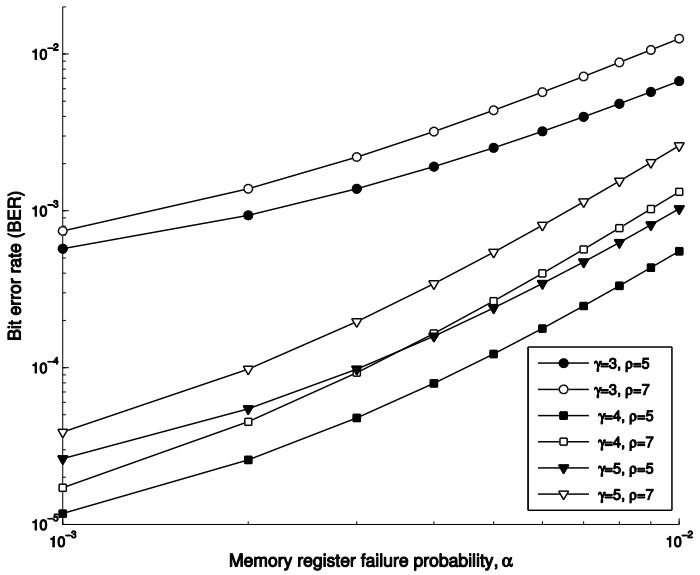


Figure 3-15: Decoders upper bounds for different (γ,ρ) classes of LDPC codes (ε=10^{-2}).

3.5 Conclusion

In this technical contribution, we evaluated the performance of QC-LDPC codes decoded by a faulty Gallager B decoder. The influence of code length, code rate and number of decoding iteration on coding system performance was analyzed. Particularly important was analysis of influence of failures in different parts of a decoder. It enables us to determine the most sensitive structures in a decoder and make them more reliable.

Due to nano-scale technologies development, assurance of the fault-tolerance became a critical issue. Using LDPC codes, as it is done in TK scheme, can significantly increase the memory reliability. We examined the transient faults influence to performance of multi-input digital gates used in TK memory architecture. The error correlation model, based on Markov chain was introduced and performance of majority logic gates and multi-input XOR gates were obtained. It was also observed that input statistics have dominant affect on faulty one-step majority logic gate performance, compared to error correlation modeled by Markov chain. Thus, simplification of faulty majority gate model was presented. The error probability at the output of multi-input XOR logic gate is determined by number of gate inputs.

Furthermore, we presented a combinatorial algorithm to calculate the exact bit error rate performance of regular LDPC codes under faulty one-step majority logic decoding. The method was applied to a several classes of LDPC codes and influence of transmitted codewords order on code performance was examined. Presented analysis represents the first step in designing low-complexity memories resistant to correlated data-dependent failures.

4. Two-Bit Bit Flipping Algorithms for LDPC Codes and Collective Error Correction

Abstract: In this technical contribution new class of bit flipping algorithms for low-density parity-check codes over the binary symmetric channel is proposed. Compared to the regular (parallel or serial) bit flipping algorithms, the proposed algorithms employ one additional bit at a variable node to represent its "strength." The introduction of this additional bit allows an increase in the guaranteed error correction capability. An additional bit is also employed at a check node to capture information which is beneficial to decoding. A framework for failure analysis and selection of two-bit bit flipping algorithms is provided. The main component of this framework is the (re)definition of trapping sets, which are the most "compact" Tanner graphs that cause decoding failures of an algorithm. A recursive procedure to enumerate trapping sets is described. This procedure is the basis for selecting a collection of algorithms that work well together. It is demonstrated that decoders which employ a properly selected group of the proposed algorithms operating in parallel can offer high speed and low error floor decoding.

4.1 Related Work on Bit-Flipping Algorithms

Among existing decoding algorithms for LDPC codes on the BSC, bit flipping algorithms are the fastest and least complex. The check node operations of these algorithms are modulo-two additions while the variable node operations are simple comparisons. Most importantly, their decoding speed does not depend on the left- and right-degree of a code. The simplicity of bit flipping algorithms also makes them amenable to analysis. Many important results on the error correction capability of bit flipping algorithms have been derived. The parallel bit flipping algorithm was shown to be capable of asymptotically correcting a linear number of errors (in the code length) for almost all codes in the regular ensemble with left degree $d_v \geq 5$ [Zyablov76]. In a later work, Sipser and Spielman used expander graph arguments to show that this algorithm and the serial bit flipping algorithm can correct a linear number of errors if the underlying Tanner graph is a good expander [Sipser96]. Recently, it was shown that regular codes with left degree $d_v = 4$ are also capable of correcting a linear number of errors under the parallel bit flipping algorithm [Burshtein08].

In recent years, numerous bit-flipping-oriented decoding algorithms have been proposed in [Nouh04] [Ngatched09], [Wu09], [Ngatched07] and [Wadayama10]. However, almost all of these algorithms require some soft information from a channel. This means that the channels assumed in these work have larger capacity than that of the BSC. A few exceptions include the probabilistic bit flipping algorithm (PBFA) proposed in [Miladinovic05]. In that algorithm, whenever the number of unsatisfied check nodes suggests that a variable (bit) node should be flipped, it is flipped with some probability $p < 1$ rather than being flipped automatically. This random nature of the algorithm slows down the decoding, which was demonstrated to be helpful in practical codes whose Tanner graphs contain cycles. The idea of slowing down the decoding can also be found in a bit flipping algorithm proposed in [Chan98]. This algorithm, which is used on the additive white Gaussian noise channel (AWGNC), requires a certain number of decoding iterations between two possible flips of a variable node.

4.2 The Class of TBF Algorithms

The error performance of bit flipping algorithms is typically inferior compared to hard-decoding message passing algorithms such as the Gallager A/B algorithm. The weakness of bit flipping decoding is especially visible for 3-left-regular codes for which the guaranteed error correction capability is upper-bounded by $\lceil g/4 - 1 \rceil$, where *g* is the girth of the Tanner graph representation of a code [Chilappagari10].

In this technical contribution we proposed the class of algorithms, in which an additional bit is introduced to represent the strength of a variable node. Given a combination of satisfied and unsatisfied check nodes, a decoding algorithm may reduce the strength of a variable node before flipping it. An additional bit is also introduced at a check node to indicate its reliability. We call the proposed algorithms two-bit bit flipping (TBF) algorithms. We next describe the principles of TBF algorithms.

Let *C* denote LDPC code defined by null space of **H**, a $m \times n$ parity check matrix of *C*. Let codeword **x** be transmitted through BSC. Consider an iterative decoder which receives binary sequence $\mathbf{y} = (y_1, y_2, \dots, y_{n,})$ and let $\hat{\mathbf{x}}^l = (\hat{x}_1^l, \hat{x}_2^l, \dots, \hat{x}_n^l)$ be the decision vector after the l-th iteration, where l is a positive integer. Let us assign a variable node to one out of four states. Specifically, in addition to the hard decision, a variable node *v* is also either a strong variable node or a weak variable node. We use 0_s, 1_s, 0_w and 1_w to denote the state of a strong zero, strong one, weak zero and weak one variable node, respectively. The set of possible states of a variable node is denoted by A_v. Let $\mathbf{w}^l = (w_1^l, w_2^l, \dots, w_n^l)$ be a vector in A_v^n such that w_v^l gives the state of variable node *v* at the end of the l-th iteration. The state w_v^0 of a variable node *v* is initialized to Δ_v^0 if $y_v = 0$ and to Δ_v^1 if $y_v = 1$, where $(\Delta_v^0, \Delta_v^1) = \{(0_s, 1_s), (0_w, 1_w)\}$.

Similarly, the extra bit can be added in check node processing, indicating the state of the parity check node, which is defined as follows.

Definition 4.1: A satisfied check node is called previously satisfied (previously unsatisfied) if it was satisfied (unsatisfied) in the previous decoding iteration, otherwise it is called newly satisfied (newly unsatisfied).

We use 0_p, 1_n, 0_p and 1_n to denote the states of a previously satisfied, a newly satisfied, a previously unsatisfied and a newly unsatisfied check node, respectively. The set of possible states of a check node is denoted by A_c. Let $\mathbf{z}^l = (z_1^l, z_2^l, \dots, z_m^l)$ be a vector in A_c^m such that z_c^l gives the state of a check *c* at the beginning of l-th iteration. Let $\mathbf{s}^l = (s_1^l, s_2^l, \dots, s_m^l)$ be a syndrome calculated at the end of l-th iteration. Thus, we have $z_c^{l+1} = \Phi(s_c^{l-1}, s_c^l)$, where check node update function $\Phi: \{0,1\}^2 \to A_c$ is defined as follows: $\Phi(0,0) = 0_p$, $\Phi(0,1) = 1_n$, $\Phi(1,0) = 0_n$ and $\Phi(1,1) = 1_p$. The state z_c^1 of a check node *c* is initialized to Δ_c^0 if $s_c^0 = 0$ and to Δ_c^1 if $s_c^0 = 1$, where $(\Delta_c^0, \Delta_c^1) = \{(0_p, 1_p), (0_n, 1_n)\}$.

Let $\chi_{0_p}^l, \chi_{0_n}^l, \chi_{1_p}^l, \chi_{1_n}^l$ be the number of previously satisfied, newly satisfied, previously unsatisfied and newly unsatisfied check nodes that are connected to a variable node *v* at the beginning of the l-th iteration, respectively. Let Ξ_{d_v} denote the set of all ordered 4-tuples $(\xi_1, \xi_2, \xi_3, \xi_4)$ such that $\xi_i \in \mathbb{N}$ and $\sum_i \xi_i = d_v$, where d_v like in the previous sections denotes weight of variable node *v*. We next define a principle of TBF algorithm.

Definition 4.2: A TBF algorithm, denoted as $\mathcal{F}(f, l_{\mathcal{F}}^m, \Delta_v, \Delta_c)$, iteratively updates $\mathbf{z}^l$ and $\mathbf{w}^l$ until all check nodes are satisfied or a maximal number of iteration $l_{\mathcal{F}}^m$ is reached. The check node update function is the previously defined function $\Phi: \{0,1\}^2 \to A_c$, while variable node update is specified by a function $f: A_v \times \Xi_{d_v} \to A_v$. The function f must satisfy the following conditions: (1) *Symmetry*: f must be symmetric with respect to 0 and 1 in the sense that if $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3, \xi_4) \in \Xi_{d_v}$ and $\left(w_{v_1}^l, w_{v_2}^l\right) \in \{(0_s, 1_s), (0_w, 1_w)\}$, then the following are true: (a)$f\left(w_{v_1}^l, \boldsymbol{\xi}\right) = 0_s \Leftrightarrow f\left(w_{v_2}^l, \boldsymbol{\xi}\right) = 1_s$, (b) $\left(w_{v_1}^l, \boldsymbol{\xi}\right) = 0_w \Leftrightarrow f\left(w_{v_2}^l, \boldsymbol{\xi}\right) = 1_w$, (c) $\left(w_{v_1}^l, \boldsymbol{\xi}\right) = 1_s \Leftrightarrow f\left(w_{v_2}^l, \boldsymbol{\xi}\right) = 0_s$ and (d) $\left(w_{v_1}^l, \boldsymbol{\xi}\right) = 1_w \Leftrightarrow f\left(w_{v_2}^l, \boldsymbol{\xi}\right) = 0_w$; (2) *Irreducibility*: Every state of a variable node must be reachable from every other state in a finite number of iterations.

4.3 Construction of Trapping Set Profile

The purpose of trapping sets analysis is identification of structures in Tanner graph which lead to failures of decoding process. Thus, we first introduce the definition of failures of a TBF algorithm.

Definition 4.3: Consider a Tanner graph *G* and a TBF algorithm $\mathcal{F}(f, l_{\mathcal{F}}^m, \Delta_v, \Delta_c)$. Let V_e denote the set of variable nodes that are initially corrupt and let *J* denote the induced subgraph on V_e (induced subgraph on V_e contains set of variable nodes V_e and all parity check nodes connected to nodes from V_e). If under the algorithm $\mathcal{F}$, decoding fails after $l_{\mathcal{F}}^m$ iterations, then we say that $\mathcal{F}$ fails because of the subgraph *J* of *G*.

Let us assume that during the transmission of the codeword **x**, the BSC makes exactly *t* errors. Denote by $\mathfrak{I}$ the set of all d_v-left-regular Tanner graphs with *t* variable nodes. It is clear that the induced subgraph on the set of initially corrupt variable nodes is isomorphic to a graph in $\mathfrak{I}$. Let *I* be a Tanner graph in $\mathfrak{I}$ and let $\mathcal{E}_I(\mathcal{F})$ denote the set of Tanner graphs containing a subgraph *J* isomorphic to *I* such that $\mathcal{F}$ fails because of *J*. The following facts follow: (1) If *C* is represented by $G \in \bigcup_{I \in \mathfrak{I}} \mathcal{E}_I(\mathcal{F})$, then there exist some weight-*t* error patterns which algorithm $\mathcal{F}$ fails to correct. (2) If, on the other hand, $G \notin \bigcup_{I \in \mathfrak{I}} \mathcal{E}_I(\mathcal{F})$, then algorithm $\mathcal{F}$ is capable of correcting any weight-*t* error patterns. Although these facts are simple, they are important elements in our analysis. However, the set of Tanner graphs $\mathcal{E}_I(\mathcal{F})$ is undeniably too general to be useful. Hence, we will focus on a subset $\mathcal{E}_I^r(\mathcal{F})$ of $\mathcal{E}_I(\mathcal{F})$ formulated as follows.

Definition 4.4: Consider a Tanner graph $S_1 \in \mathcal{E}_I(\mathcal{F})$ such that $\mathcal{F}$ fails because of the subgraph J_1 of S_1. Then, $\mathcal{E}_I^r(\mathcal{F})$ consists of all S_1 for which there does not exist $S_2 \in \mathcal{E}_I(\mathcal{F})$ such that: (1) $\mathcal{F}$ fails because of some subgraph J_2 of S_2, and (2) there is an isomorphism between S_2 and a proper subgraph of S_1 under which the variable node set $V(J_2)$ is mapped into the variable node set $V(J_1)$.

Definition 4.5: If $S \in \mathcal{E}_I(\mathcal{F})$ then *S* is a trapping set of $\mathcal{F}$.*I* is called an inducing set of *S*. $\mathcal{E}_I^r(\mathcal{F})$ is called the trapping set profile with inducing set *I* of $\mathcal{F}$.

An important property of a trapping set, which enables the construction of a trapping set profile, is stated in the following proposition.

Proposition 4.1: Let $S \in \mathcal{E}_I(\mathcal{F})$ be a trapping set of $\mathcal{F}$ with inducing set *I*. Then, there exists at least one induced subgraph *J* of *S* which satisfies the following properties: (1) *J* is isomorphic to *I*, and (2) $\mathcal{F}$ fails because of *J* of *S*, and (3) Consider the decoding of $\mathcal{F}$ on *S* with *V*(*J*) being the set of initially corrupt variable nodes. Then, for any variable node $v \in V(S)$, there exist an integer l such that $0 \leq l \leq l_{\mathcal{F}}^m$ and $w_v^l \in \{1_s, 1_w\}$.

The recursive procedure for constructing a trapping set profile $\mathcal{E}_I^r(\mathcal{F})$ relies on Proposition 4.1. In particular, consider a trapping set *S* with an inducing set *I*, Proposition 4.1 states that for at least one induced subgraph *J* of *S* which is isomorphic to *I*, in the decoding $\mathcal{F}$ on *S* with *V*(*J*) being the set of initially corrupt variable nodes, every variable node in *V*(*S*) is corrupt at the end of some iteration. As a result, given the knowledge of *J*, it is possible to generate *S* by simultaneously performing decoding and adding variable nodes to *J* in *one specific manner*. Consequently, if we simultaneously perform decoding and add variable nodes to *I* in *all possible ways*, then all trapping sets with inducing set *I* can be generated. We now describe this procedure.

Let us assume that we are only interested in trapping sets with at most n_{max} variable nodes. Consider the decoding of $\mathcal{F}$ on a Tanner graph *I* with *V*(*I*) being the set of initially corrupt variable nodes. Let $n_I = |V(I)|$. If $\mathcal{F}$ fails because of *I* then $\mathcal{E}_I^r(\mathcal{F}) = \{I\}$ and we have found the trapping set profile. If $\mathcal{F}$ does not fail because of *I*, then we expand *I* by recursively adding variable nodes to *I* until a trapping set is found. During this process, we only add variable nodes that become corrupt at the end of a certain iteration.

Consider all possible bipartite graphs obtained by adding one variable node, namely v_{n_I+1}, to the graph *I* such that when the decoding is performed on these graphs with *V*(*I*) being the set of initially corrupt variable nodes, the newly added variable node is a corrupt variable node at the end of the first iteration, i.e., $w_{v_{n_I+1}}^1 \in \{1_s, 1_w\}$. Let $\mathbf{O}_I$ denote the set of such graphs. Take one graph in $\mathbf{O}_I$ and denote it by *U*. Then, there can be two different scenarios in this step: (1) $\mathcal{F}$ does not fail on the subgraph *I* of *U*. In this case, *U* is certainly not a trapping set and we put *U* in a set of Tanner graphs denoted by E_I^1. (2) $\mathcal{F}$ fails because of the subgraph *I* of *U*. In this case, *U* is a potential trapping set and a test is carried out to determine if *U* is indeed one. If *U* is not a trapping set then it is discarded. We complete the formation of E_I^1 by repeating the above step for all other graphs in $\mathbf{O}_I$.

Let us now consider a graph $U \in E_I^1$. Again, we denote by $\mathbf{O}_U$ the set of Tanner graphs obtained by adding one variable node, namely v_{n_I+2}, to the graph *U* such that when the decoding is performed on these graphs with *V*(*I*) being the set of initially corrupt variable nodes, the newly added variable node is a corrupt variable node at the end of the first iteration, i.e., $w_{v_{n_I+2}}^1 \in \{1_s, 1_w\}$. It is important to note that the addition of variable node v_{n_I+2}, which is initially correct, cannot change the fact that variable node v_{N_I+1} is also corrupt at the end of the first iteration. This is because the addition of correct variable nodes to a graph does not change the states of the existing check nodes and the decoding dynamic until the moment the newly added variable nodes get corrupted. Similar to what has been discussed before, we now take a graph in $\mathbf{O}_U$ and determine if it is a trapping set, or it is to be discarded, or it is a member of the set of Tanner graph E_I^2. By repeating this step for all other graphs in E_I^1, all graphs in E_I^2 can be enumerated. In a similar fashion, we obtain $E_I^3, E_I^4, \dots, E_I^{n_{max}-n_I}$. For the sake of convenience, we also let $E_I^0 = \{I\}$.

At this stage, we have considered one decoding iteration on *I*. It can be seen that if *S* is a trapping set with at most n_{max} variable nodes then either *S* has been found, or *S* must contain a graph in $\bigcup_{i=0}^{n_{max}-n_I-1} E_I^i$. Therefore, we proceed by expanding graphs in $E_I = \bigcup_{i=0}^{n_{max}-n_I-1} E_I^i$.

Let *K* denote a Tanner graph in $E_I = \bigcup_{i=0}^{n_{max}-n_I-1} E_I^i$. We now repeat the above graph expanding process starting from *K*. Specifically, we first obtain $\mathbf{O}_K$, which is defined as the set of all Tanner graphs obtained by adding one variable node v_{n_K+1} to the graph *K* such that when the decoding is performed on these graphs with *V*(*I*) being the set of initially corrupt variable nodes, the newly added

variable node is a corrupt variable node at the end of the second iteration, but not a corrupt variable node at the end of the first iteration, i.e., $w^1_{v_{n_K+1}} \in \{0_s, 0_w\}$ and $w^2_{v_{n_K+1}} \in \{1_s, 1_w\}$. The graphs in $\mathbf{O}_K$ that are not trapping sets are either discarded or form the set E^1_K. By iteratively adding variable nodes, we enumerate all graphs in $E^2_K, E^3_K, \ldots, E^{n_{max}-n_I}_K$.

One can see that there are two different recursive algorithms. The first algorithm enumerates graphs in $E_K = \bigcup_{i=0}^{n_{max}-n_I-1} E^i_K$ for a given graph K by recursively adding variable nodes. The second algorithm recursively calls the first algorithm to enumerate graphs in $E_K = \bigcup_{i=0}^{n_{max}-n_I-1} E^i_K$ for each graph K in $E_I = \bigcup_{i=0}^{n_{max}-n_I-1} E^i_I$. Each recursion of the second algorithm corresponds to a decoding algorithm. As a result, the trapping set profile is obtained after $l^m_{\mathcal{F}}$ recursions of the second algorithm. The pseudocodes of the two algorithms, which we call **RA1** and **RA2** can be found in [Nguyen13]. The interested readers are referred to [Nguyen13] for an example which demonstrates the operations of these recursive algorithms.

4.4 Application of the Failure Analysis in the Section of TBF Algorithms

The failure analysis presented in the previous section enables the enumeration of all relevant uncorrectable error patterns for a given TBF algorithm. Consequently, the selection of one good TBF algorithm becomes simple. Nevertheless, even if the best TBF algorithm can now be found, its error performance might not be attractive enough for certain applications. In such a scenario, because of the simplicity and high throughput of bit flipping decoding, it is natural to consider the use of multiple TBFAs operating in a complimentary manner. In the following discussion, we introduce the concept of collective error correction and emphasize on the ease of selecting complimentary TBF algorithms.

Let us consider a collection $\mathbf{A}$ of (general) iterative decoding algorithms for LDPC codes. Assume for a moment that the set of all uncorrectable error patterns for each and every algorithm in $\mathbf{A}$ is known. More precisely, in the context of LDPC codes, we assume that all the induced subgraphs on such error patterns can be enumerated for each decoding algorithm. This naturally suggests the use of a decoder $\mathbf{D}$ which employs multiple algorithms drawn from $\mathbf{A}$. The basis for this use of multiple algorithms is rather simple: If different algorithms are capable of correcting different error patterns, then a decoder employing a set of properly selected algorithms can achieve provably better error performance than any single-algorithm decoder. Although the above assumption is not valid for most iterative algorithms, it is valid for the TBF algorithms defined in this section. This is because the notion of trapping set of a TBF algorithm helps determine whether or not an arbitrary error pattern is correctable. The concept and explicit construction of trapping set profiles allow rigorous selections of multiple algorithms which can collectively correct a fixed number of errors with high probability. In particular, a decoder employing algorithms with diverse trapping set profiles is capable of correcting a wide range of error configurations, hence possesses desirable error correction capability. Although the selection of algorithms can be code independent, any knowledge on the Tanner graph can be utilized to refine the selection.

The inputs to the process of selecting a group of TBF algorithms are: (1) A large collection $\mathbf{A}$ of TBF algorithms. A can be the set of all possible algorithms or it can be a subset of algorithms that satisfies certain constraints. (2) A set of Tanner graphs with a small number of variable nodes. These are all possible subgraphs induced on the set of initially corrupt variable nodes. (3) Optional knowledge on the Tanner graph on which the decoder operates. The main element of the selection process is the

enumeration of trapping sets for all input algorithms to generate their trapping set profiles. Once the trapping set profiles of all input algorithms are generated, they will be used as inputs to the algorithm selector. The algorithm selector outputs a set of algorithms drawn from $\mathbf{A}$ with diverse trapping set profiles. By cycling though these algorithms, or operating them in parallel, it is possible to overcome the most commonly-occurring error configurations, thereby improving the error performance.

Let $n_{I,\mathcal{F}}^{min}$ be the smallest number of variable nodes of Tanner graphs in $\mathcal{E}_I^r(\mathcal{F})$. Then, one should select an algorithm $\mathcal{F}$ such that $n_{I,\mathcal{F}}^{min}$ is maximized. To justify this selection criterion, we rely on results in extremal graph theory [McKay10].

We now consider the problem of selecting multiple algorithms. The basis for this selection is that one should select good individual algorithms with diverse trapping set profiles. In this technical contribution, we only consider a decoder D with algorithms $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_p$ operating in parallel, i.e., the received vector of the channel is the input vector for all algorithms. Note that one can also use trapping set profiles to select algorithms that operate in serial, i.e., the output from one algorithm is the input to another. For a decoder D that employs parallel algorithms, the concept of trapping sets and trapping set profiles can be defined in the same manner as trapping sets and trapping set profiles for a single TBF algorithm. One can easily modify the recursive procedures given in previous section to generate trapping set profiles of the decoder D. Then, D can be designed with the same criterion discussed previously.

4.5 Numerical Results

In this section, we give examples of selecting TBF algorithms and demonstrate the frame error rate (FER) performance of decoders employing these algorithms. For simplicity, we assume the use of 3-left-regular codes with girth $g = 8$. We also let $\Delta_v = (0_s, 1_s)$, $\Delta_c = (0_p, 1_p)$ and $l_{\mathcal{F}}^m = 30$ for all algorithms. Our algorithms are compared with the Gallager A/B algorithm, the min-sum algorithm and the SPA. In these comparisons, the Gallager A/B algorithm, the min-sum algorithm and the SPA iterate for a maximum of 100 iterations. In the SPA, messages have floating-point precision. The comparisons are made on two codes: (1) The popular (155,64) Tanner code [Tanner01], denoted by C_1. This code has $d_v = 3$, $d_c = 5$, rate $R = 0.4129$ and minimum distance $d_{min} = 20$. (2) A quasi-cyclic code of length $n = 732$, rate $R = 0.7527$ and minimum distance $d_{min} = 12$ [Nguyen12]. We denote this code by C_2. As previously mentioned, only a subset $\mathbf{A}_r$ of TBF algorithms should be considered. The constraints on TBF algorithms that define $\mathbf{A}_r$ are constraints on the images of the variable node update functions f. We are considering a set of 41,472,000 algorithms that satisfy a specific set of constraints. All the algorithms given in this task and the constraints on them can be found at [Vasic13-a].

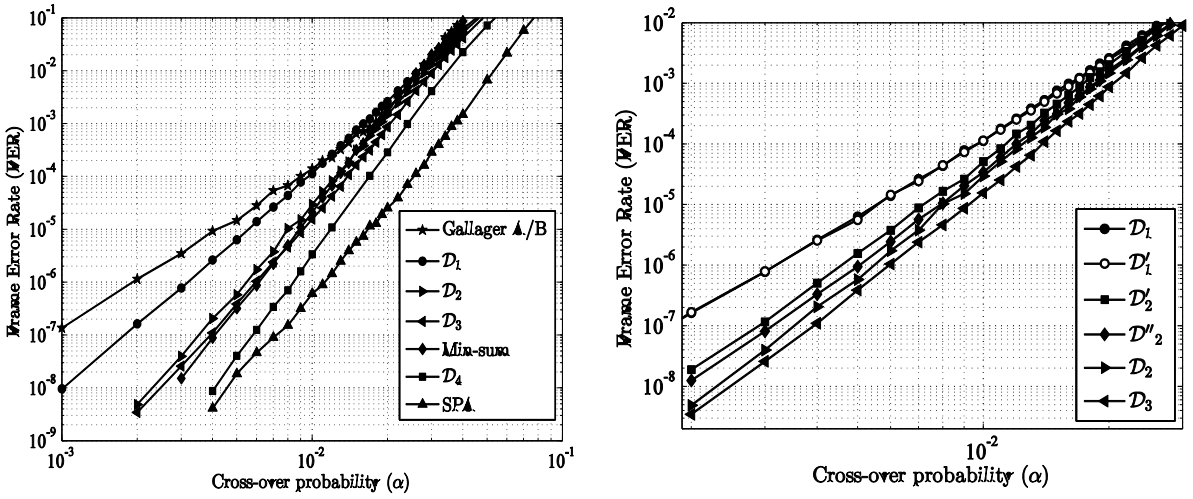
It is expected that a single algorithm can guarantee the correction of 3 errors in a girth-8 3-left-regular LDPC code. Let $\mathfrak{I}_3$ be the set of all Tanner graphs with girth $g = 8$ and three variable nodes. We generate the trapping set profiles $\mathcal{E}_I^r(\mathcal{F})$ for all $\mathcal{F} \in \mathbf{A}_r$ and all $I \in \mathfrak{I}_3$. There is a set $\mathbf{A}_r^3$ of 358,851 algorithms for which $\mathcal{E}_I^r(\mathcal{F}) = \emptyset$ for all $I \in \mathfrak{I}_3$. Such an algorithm can guarantee to correct all weight-three error patterns. Our next step is to select an algorithm which is most expected to correct a weight-four error pattern. Let $\mathfrak{I}_4$ be the set of all Tanner graphs with girth $g \geq 8$ and four variable nodes. Then, $|\mathfrak{I}_4| = 10$. We generate the trapping set profiles $\mathcal{E}_I^r(\mathcal{F})$ for all $\mathcal{F} \in \mathbf{A}_r$ and all $I \in \mathfrak{I}_4$. The largest among the smallest sizes of trapping sets in $\mathcal{E}_I^r(\mathcal{F})$ for all $\mathcal{F} \in \mathbf{A}_r$ are

$(10,8,8,7,7,\infty,\infty,\infty,6,4)$. There are six algorithms, denoted as $\mathcal{F}_{A1},\mathcal{F}_{A2},\ldots,\mathcal{F}_{A6}$ for which the smallest sizes of trapping sets in $\mathcal{E}_I^r(\mathcal{F})$ are $(10,8,8,5,6,\infty,\infty,\infty,5,4)$. The smallest sizes of trapping sets in $\mathcal{E}_I^r(\mathcal{F})$ for these algorithms overlap the most with $\max(n_{I_1}^{min},n_{I_2}^{min},\ldots,n_{I_{10}}^{min})$. Hence, these algorithms have the highest probability to correct a random error pattern of weight-four. Let D_1 denote a decoder which uses $\mathcal{F}_{A1}$. The FER performance of D_1 on C_1 and C_2 are shown in Fig. 4-1(a) and Fig. 4-2, respectively. It can be seen that D_1, which operates with a maximum of only 30 iterations, outperforms the Gallager A/B algorithm in both codes and outperform the min-sum algorithm on C_2, which is a higher rate code.

Let D_1' denote a decoder which uses $\mathcal{F}_{A1},\mathcal{F}_{A2},\ldots,\mathcal{F}_{A6}$ in parallel. Fig. 4-1(b) shows the performance of D_1' in comparison with D_1. One can observe that there is no gain in the FER performance of D_1' . Simply using in parallel a group of good algorithms does not necessarily result in better error correction capability. We shall now give an example on how to properly selecting a collection of algorithms. Assume that $\mathcal{F}_{A1}$ is already in use. We would like to select other TBF algorithms to complement $\mathcal{F}_{A1}$. In particular, we need to select more TBF algorithms to complement $\mathcal{F}_{A1}$ in correcting weight-four error patterns. From the set of trapping set profiles for all algorithms in $\mathbf{A}_r$, we can select three algorithms, denoted as $\mathcal{F}_{B1},\mathcal{F}_{B2}$.and $\mathcal{F}_{B3}$. The smallest sizes of trapping sets in $\mathcal{E}_I^r(\mathcal{F})$ for these algorithms are $(8,8,7,8,8,8,9,\infty,8,4)$, $(9,8,8,6,6,\infty,11,\infty,9,4)$ and $(4,4,4,4,4,4,4,4,4,\infty)$, respectively. Note that these three algorithms are not capable of correcting all weight-three error patterns. Let D_2' denote a decoder which uses $\mathcal{F}_{A1}$ and $\mathcal{F}_{B1}$ in parallel. Let D_2'' denote a decoder which uses $\mathcal{F}_{A1},\mathcal{F}_{B1},\mathcal{F}_{B2}$ in parallel. Also let D_2 denote a decoder which uses $\mathcal{F}_{A1},\mathcal{F}_{B1},\mathcal{F}_{B2}$.and $\mathcal{F}_{B3}$ in parallel. The FER performance of D_2, D_2' 2 and D_2'' on C_1 in comparison with D_1 are shown in Fig. 4-1 (b). The FER performance of D_2 on C_1 and C_2 are also shown in Fig. 4-1(a) and Fig. 4-2, respectively. One can observe the gradual improvement in the FER performance when more algorithms are used. As previously mentioned, knowledge on the Tanner graph can be useful in the selection of algorithms. By using the method of searching for subgraphs proposed in [Nguyen12], one can conclude that the Tanner graph of C_1 is free of some subgraphs included in the trapping set profiles. After deleting irrelevant trapping sets from the trapping set profiles, we can select a set of four TBF algorithms that are used by a decoder D_3. The FER performance of D_3 on C_1 and C_2 are shown in Fig. 4-1(a) and Fig. 4-2, respectively. D_3's performance on C_1 is also shown in Fig. 4-1(b) to facilitate the comparison. One can observe that D_3 outperforms D_2 on C_1. Note that although D_3 also outperforms D_2 on C_2, this fact is not expected to be typical.

Finally, we construct all trapping set profiles with inducing sets containing four, five and six variable nodes for each algorithm in $\mathbf{A}_r$. Note that there are 10 possible inducing sets (Tanner graphs with girth $g \geq 8$) containing four variable nodes, 24 possible inducing sets containing five variable nodes and 57 possible inducing sets containing six variable nodes. Hence, for each algorithm, we construct a total of 92 trapping set profiles. From the trapping set profiles of all algorithms, we select a collection of 35 algorithms. Then, we simulate the performance of a decoder D_4 which employs these algorithms in parallel. The maximum total number of iterations of D_4 is $35 \times 30 = 1050$. Fig. 4-1(a) shows the FER performance of D_4 on C_1. It can be seen that the FER performance of D_4 approaches (and might surpass) that of the SPA in the error floor region. It is also important to note that D_4 can correct any error pattern up to weight 5 on C_1. Fig. 4-2 also shows the FER performance of D_4 on C_2. It can be seen that the slope of the FER curve of D_4 in the error floor region is higher than that of the SPA. This indicates that the FER performance of D_4 might eventually surpass that of the SPA. Finally, we remark that the slope of the FER curve of D_4 in the error floor region is between 5 and 6, which

indicates that D_4 can correct error patterns of weight 4 and 5 with high probability. This also agrees with the fact that in our simulation, no weight-four error pattern that leads to decoding failure of D_4 was observed.



a) b)

Figure 4-1: FER performance on code C_1.

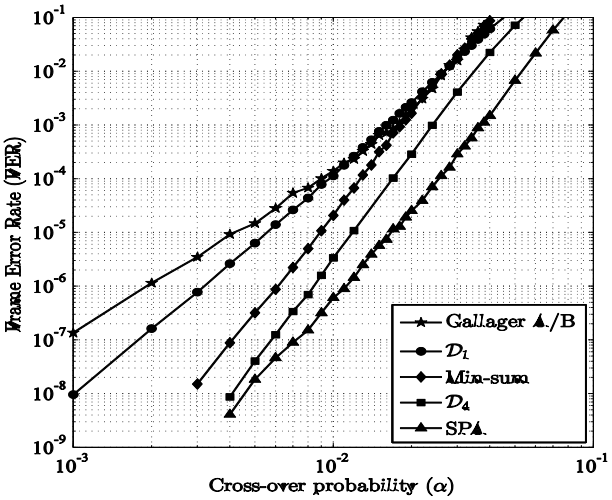


Figure 4-2: FER performance on code C_2.

4.6 Conclusion

We proposed a novel class of bit flipping algorithms for LDPC codes. More importantly, we gave a framework for the analysis and selection of algorithms that can together correct a fixed number of errors with high probability. Since in TBF algorithms variable nodes take more than two states, it would be interesting to consider the use of the proposed algorithms on channels.

Good correcting abilities of these low complexity decoders makes them potentially useful for construction of reliable memories. Our next step is directed to analysis of presented decoders under unreliable computation.

5. Failures and Error-Floors of Iterative Decoders

Abstract: In this technical contribution we analyze iterative decoder failures and approaches for increasing LDPC codes performance in error-floor region. Based on the progressive constructing of Tanner graphs by avoiding trapping sets we construct a LDPC code with same row and column weights, but superior error-floor performance compared to well known Tanner code. Additionally, we propose a novel class of finite alphabet iterative decoders (FAIDs) named decimation-enhanced FAIDs with fast convergence and guaranteed error correction capabilities.

5.1 Overview of Decoding Failures and Error-Floor Analysis

The properties of LDPC codes under different decoding algorithms have been analyzed by studying ensembles of codes in the limit of code length approaching infinity, or equivalently under the assumption that the underlying Tanner graph is a tree. Unfortunately, the methods to analyze ensembles of codes are of limited use for the performance analysis of a given finite length code especially at error-floor region when performance of particular codes diverge. Thus, error-floor analysis is restricted to a particular code. In this section we define failures of iterative decoders and describe influence of trapping sets on decoder failures.

Analytical characterization of the performance of a code requires an understanding of the conditions that lead to the failure of the decoder under question. A decoder is said to have failed when the output of the decoder is not a codeword. Note that it is possible that the decoder finds a codeword different from the transmitted codeword and this is generally referred to as a decoding error. The decoding failures and errors are a function of the code, the decoder and the channel. In the case of iterative message algorithms, the decoders operate on the Tanner graph of the code and consequently the failures and errors are also a function of the topological structure of the Tanner graph. They also depend on the message update rules, on the message passing schedule and the implementation aspects such as message precision.

To better understand the error floor phenomenon, we consider as an example, the estimation of the FER of (N,K) LDPC code on the BSC. Denote c_k as the number of configurations of received bits for which k channel errors lead to codeword (frame) error for an iterative decoder. The FER is given by:

$$FER(\alpha) = \sum_{k=i}^{N} c_k \alpha^k (1-\alpha)^{N-k}, \tag{5-1}$$

where α denote BSC crossover probability and i is the minimal number of channel errors that can lead to a decoding error. On semilog scale FER is given by

$$\begin{aligned} \log\big(FER(\alpha)\big) &= \log\left(\sum_{k=i}^{N} c_k \alpha^k (1-\alpha)^{N-k}\right) \\ &= \log(c_i) + i\log(\alpha) + (N-i)\log(1-\alpha) + \cdots \\ &+ \log\left(1 + \frac{c_{i+1}}{c_i}\alpha(1-\alpha)^{-1} + \cdots + \frac{c_N}{c_i}\alpha^{(N-i)}(1-\alpha)^{-i}\right) \end{aligned} \tag{5-2}$$

For a small α, the expression is dominated by the first two terms thereby reducing to

$$\log\big(FER(\alpha)\big) = \log(c_i) + i\log(\alpha). \tag{5-3}$$

From the above expressions, we can see that on a log(*FER*) vs log(α) scale, the slope of the error floor is governed by the parameter *i*. It is evident from previous discussions that the main element in the analysis and estimation of error floor is the determination of smallest-weight error patterns that lead to decoding failures. We know focus on the combinatorial characterization of decoding failures utilizing the notion of trapping sets.

We assume that the iterative decoder performs a finite number of iterations, denoted as *D*. To define the notion of trapping sets, we shall assume that *D* is as large as necessary. Denote by **x** the transmitted codeword. Consider an iterative decoder and let $\hat{\mathbf{x}}^l = (\hat{x}_1^l, \hat{x}_1^l, \ldots \hat{x}_N^l)$ be the decision vector after the l-th iteration. We say that a variable node *v* is eventually correct if there exists a positive integer l_c such that for all l with $l_c \leq l$, $\hat{x}_v^l = x_v$. Then, trapping sets are defined as follows.

Definition 5.1 ([Richardson03]): A trapping set for an iterative decoding algorithm is a non-empty set of variable nodes in a Tanner graph *G* that are not eventually correct. Such a set of variable nodes, say **T**, is called an (*a,b*) trapping set if it contains *a* variable nodes and the subgraph induced by these variable nodes has *b* odd-degree check nodes.

Depending on the context, a trapping set can be understood as a set of variable nodes in a given code with a specified induced subgraph or it can be understood as a specific subgraph independent of a code. To differentiate between these two cases, we use the letter $\mathbf{T}$ to denote a set of variable nodes in a code and use the letter Π to denote a type of trapping set which corresponds to a specific subgraph.

The following definition gives the topological relations among trapping sets which represent bases of trapping sets ontology (TSO).

Definition 5.2: A trapping set Π_2 is a successor of a trapping set Π_1 if there exists a proper subset of variable nodes of Π_2 that induce a subgraph isomorphic to the induced subgraph of Π_1. If Π_2 is a successor of Π_1 then Π_1 is a predecessor of Π_2. Furthermore, Π_2 is a direct successor of Π_1 if it does not have a predecessor Π_3 which is a successor of Π_1.

Remark: A trapping set Π can have multiple, incomparable predecessors.

If Π_2 is a successor of Π_1, then the topological relation between Π_1 and Π_2 is solely dictated by the topological properties of their subgraphs. In the Tanner graph of a code *C*, the presence of a trapping set $\mathbf{T}_1$ of type Π_1 does not indicate the presence of a trapping set $\mathbf{T}_2$ of type Π_2. If $\mathbf{T}_1$ is indeed a subset of a trapping set $\mathbf{T}_2$ in the Tanner graph of code *C*, then we say that $\mathbf{T}_1$ generates $\mathbf{T}_2$, otherwise we say that $\mathbf{T}_1$ does not generate $\mathbf{T}_2$.

The web page [Vasic13-b] contains a complete description of trapping sets for column weight three codes up to eight variable nodes, including their topological structure and parent-child relationships.

Chilappagari *et al.* [Chilappagari06] gave a method for FER estimation which consists of the following three main steps.

1) Identifying the relevant classes of trapping sets $\Pi_1, \Pi_2, \ldots$.

2) Calculating the number of trapping sets of each class, i.e., the number of trapping sets $\mathbf{T}_1$ of type Π_1, the number of trapping sets $\mathbf{T}_2$ of type Π_2 and so on.

3) Calculating the contribution of each class of trapping set, taking into account the topological relationship among them, as well as whether one trapping set generates another in the Tanner graph.

Assume that in the Tanner graph that corresponds to a parity-check matrix **H**, there are $|\Pi|$ trapping sets of type Π. Let *m* be the critical number of type Π trapping sets. For simplicity, we also assume that corresponding to one trapping set **T** of type Π , there is exactly one inducing set of cardinality *m*. The contribution of a class of trapping sets Π , $\Pr\{\Pi\}$, to the FER is calculated by:

$$\Pr\{\Pi\} = \sum_{r=m}^{M} \Pr\{\Pi|\mathrm{r}-\mathrm{errors}\}\Pr\{\mathrm{r}-\mathrm{errors}\}, \tag{5-4}$$

where $\Pr\{\Pi|\mathrm{r}-\mathrm{errors}\}$ is probability that the decoder ends in a trapping set of the class Π, given that the channel introduced *r* errors while $\Pr\{\mathrm{r}-\mathrm{errors}\}$ is the probability that the channel introduced r errors. Thus, we have

$$\Pr\{\Pi|\mathrm{r}-\mathrm{errors}\} = \frac{|\Pi|\binom{r}{m}}{\binom{N}{m}} \tag{5-5}$$

and

$$\Pr\{\mathrm{r}-\mathrm{errors}\} = \binom{N}{r}\alpha^r(1-\alpha)^{N-r}. \tag{5-6}$$

M is the maximum number of errors which can end up in the trapping set. If more than *M* errors are introduced by the channel, the decoder still fails to correct these errors but in this case the decoder does not end in the trapping set.

5.2 Constructing Tanner graphs by Avoiding Trapping Sets

From the previous section, it is clear that the error floor problem arises due to the suboptimality of iterative decoding on loopy graphs. For this reason, to alleviate the effect of error floor, it is natural to consider two approaches:

1) Find a Tanner graph on which a given decoding algorithm is least susceptible to (error floor) failures.

2) Based on error floor analysis on a given Tanner graph, adjust a given decoding algorithm/decoder to achieve better error floor performance.

In this section we present approach for progressive constructing of Tanner graphs. Unlike the well known PEG (*Progressive Edge-Growth*) algorithm which constructs a Tanner graph with a given set of parameters, in the progressive construction of Tanner graphs free of small trapping sets, the length is usually not pre-specified. The construction is based on a check and select-or-disregard procedure. For example, in the progressive construction of structured regular LDPC codes described in [Nguyen12], the Tanner graph of the code is built in stages, where ρ is the row weight of parity check matrix **H**. ρ is not pre-specified, and a code is constructed with the goal of making the rate as high as possible. At each stage, a set of new variable nodes are introduced that are initially not connected to the check nodes of the Tanner graph. Blocks of edges are then added to connect the new variable nodes and the check nodes. After a block of edges is tentatively added, the Tanner graph is checked to see if it contains any undesired trapping set. If it does, then that block of edges is removed and replaced by a different block. The algorithm proceeds until no block of edges can be added without introducing undesired trapping sets to the Tanner graph. When constructing a Tanner graph by progressively adding variable nodes, the under-constructing Tanner graph is checked to see if it contains certain trapping sets. This can only be done by exhaustively searching for trapping sets.

An efficient search of the Tanner graph for trapping sets relies on the topological relations defined in the trapping set databases TSO and/or carefully analyzing their induced subgraphs. It is easy to show that the induced subgraph of every trapping set contains at least one cycle. Therefore, the search for trapping sets begins with enumerating cycles up to a certain length. Also a cycle with a variable nodes is an (a,a) trapping set. After the cycles have been enumerated, they will be used in the search for larger trapping sets. A larger trapping set can be found in a Tanner graph by expanding a smaller trapping set. More precisely, given a trapping set $\mathbf{T}_1$ of type Π_1 in the Tanner graph of a code C, our techniques search for a set of variable nodes such that the union of this set with $\mathbf{T}_1$ forms a trapping set $\mathbf{T}_2$ of type Π_2, where Π_2 is a successor of Π_1. These techniques are sufficient to efficiently search for a large number of trapping sets in the TSO.

Let us now give a general rationale for deciding which trapping sets should be forbidden in the Tanner graph of a code. To facilitate the discussion, we assume that the forbidden trapping sets are drawn from the TSO. It is clear that if a predecessor trapping set is not present in a Tanner graph, then neither are its successors. Since the size of a predecessor trapping set is always smaller than the size of its successors, a code should be constructed so that it contains as few small predecessor trapping sets as possible. However, forbidding smaller trapping sets usually imposes stricter constraints on the Tanner graph, resulting in a large rate penalty. This trade-off between the rate and the choice of forbidden trapping sets is also a trade-off between the rate and the error floor performance. While an explicit formulation of this trade-off is difficult, a good choice of forbidden trapping sets requires the analysis of decoder failures to reveal the relative harmfulness of trapping sets. It has been pointed out that for the BSC, the slope of the FER curve in the error floor region depends on the size of the smallest error patterns uncorrectable by the decoder [Nguyen12]. We therefore introduce the notion of the relative harmfulness of trapping sets in a general setting as follows.

Relative Harmfulness: Assume that under a given decoding algorithm, a code is capable of correcting any error pattern of weight ϑ but fails to correct some error patterns of weight $\vartheta + 1$. If the failures of the decoders on error patterns of weight $\vartheta + 1$ are due to the presence of (a_1,b_1) trapping sets of type Π_1, then Π_1 is the most harmful trapping set. Let us now assume that a code is constructed so that it does not contain Π_1 trapping sets and is capable of correcting any error pattern of weight $\vartheta + 1$. If the presence of (a_2,b_2) trapping sets of type Π_2 leads to decoding failure on some error patterns of weight $\vartheta + 1$, then Π_2 is the second most harmful trapping set. The relative harmfulness of other trapping sets are determined in this manner.

Remarks: According to the above discussion, a smaller trapping set might not necessarily be more harmful than a larger one. Besides, for two trapping sets with the same number of variable nodes but with different number of odd degree check nodes, the one with the smaller number of odd degree check nodes might not necessarily be more harmful.

We next investigate the importance of avoiding trapping sets by comparing error-floor performance of code constructed using presented technique and well known Tanner code [Tanner01]. Tanner code is a (3,5)-regular LDPC code with girth g = 8 and minimum distance $d_{\min}$ = 20. Its Tanner graph contains (5,3) trapping sets. Since the critical number of (5,3) trapping sets is three, the code cannot correct three errors under the Gallager A/B algorithm on the BSC. We used the construction technique described above to construct a different code with the same length, column weight and row weight. Let C_1 denote this code. It is a (155,64) LDPC code with girth g = 8 and minimum distance $d_{\min}$ = 12. The Tanner graph of C_1 contains no (5,3) trapping sets. Therefore, C_1 is capable of

correcting any three-error pattern under the Gallager A/B algorithm on the BSC. The FER performance of C_1 under the Gallager A/B algorithm for a maximum of 100 iterations is shown in Fig. 5-1. The FER performance of the Tanner code is also shown for comparison. As suggested in Fig. 5-1, the minimum distance does not matter in the error floor region.

Constructed code C_1 has the same coding rate as Tanner code, which means that has the same decoding complexity. Thus, we expect that C_1 will also be superior under unreliable computation decoding.

In our further investigation we will use findings described in this section as direction for assuring reliable storage of information.

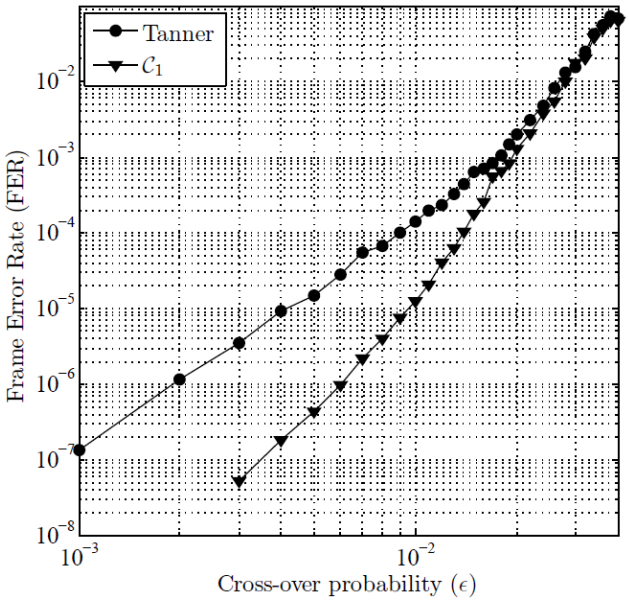


Figure 5-1: Frame error rate performance of the Tanner code and code C_1 under the Gallager A/B algorithm on the BSC with maximum of 100 iterations.

5.3 Decimation-enhanced Finite Alphabet Iterative Decoders with Provable Guaranteed Error-Correction for LDPC codes

This activity in WP4 is still under development and we are in the process of finalizing a journal paper submission related to this topic. We provide in this section an explanation of the problem that we address together with a summary of our findings and results. All the technical details can be found in appendix-A, which is composed of the self-contained description of our approach based on decimation-enhanced FAID.

While it has been theoretically established by Taylor [Taylor68-A] and Kuznetsov [Kuznetsov73] that iterative LDPC decoders need to be employed for fault-tolerant memories in order to achieve a non-zero capacity asymptotically with the length of the code, for practical systems which use finite-length codes, the decoders are known to suffer from the error floor problem. The error floor phenomenon is an abrupt degradation in the error-rate performance of the code which occurs when the required error-rates are very low, and on the binary symmetric channel (BSC), the slope of the error floor is governed by the guaranteed error-correction capability [Ivkovic06-IT] of the code. Moreover, preliminary investigations on the design of fault-tolerant memories [Vasic07] have revealed that it is

crucial for the employed LDPC decoders to be able to correct errors in a small number of iterations in order to prevent the phenomenon of error propagation along the decoding iteration. This phenomenon occurs when errors associated with the probabilistic gate failures are introduced with each additional decoding iteration thereby resulting in a loss of memory efficiency. Hence, the design of LDPC decoders that can achieve a guaranteed error-correction in the fewest possible iterations is of paramount importance in order to be able to realize practical fault-tolerant memories.

So far, preliminary works on the design and analysis of fault-tolerant memories that have considered iterative LDPC decoders include both the bit flipping decoders such as [Chilappagari07], as well as the message-passing decoders such as one-step majority logic decoder [Chilappagari06] and the Gallager-B decoder which the original TK scheme can be regarded as. Message-passing decoders are in general more complex to analyze than bit-flipping decoders, but can also achieve a higher reliability. Although the above considered decoders are simple hard-decision decoders that are amenable to analysis, their error-correction capability for finite-length codes is weak. Moreover, the finite-length analysis of fault-tolerant memories was only considered for the one-step majority logic decoder [Chilappagari06] which uses only a single decoding iteration. In order to reduce the redundancy rates required for fault-tolerant memories, it would be desirable to consider more powerful message-passing decoders that are capable of correcting a higher number of errors with only a marginal increase in complexity, and also allow the decoder analysis to consider more than one iteration. Recently a new class of finite-precision message-passing decoders called *finite alphabet iterative decoders* (FAIDs) were proposed in [Planjery13] wherein the messages belong to a finite alphabet. These decoders are capable of surpassing the belief propagation (BP) decoder in the error floor performance at much lower complexity and requiring only a small precision (as small as three bits) compared to the BP algorithm to represent the messages. Therefore, FAIDs are strong candidates to consider for application in fault-tolerant memories especially if they can be analyzed for finite number of iterations in terms of guaranteed error-correction.

However, the finite-length analysis of any message-passing decoder (other than the Gallager-A/B decoder) for a finite number of iterations still remains a challenge, and the guaranteed error-correction of finite-length LDPC codes under message-passing decoding remains largely unknown. This is because the message-passing decoder typically requires many iterations to correct a certain number of errors, and the dynamics of message-passing becomes too complex to analyze beyond even a few iterations due to the exponential growth in the number of nodes with the number of iterations in the corresponding computation trees of the decoder. This is also true for the FAIDs for which only numerical results have been provided in [Planjery13] to demonstrate their increased guaranteed error-correction. The only results known on guaranteed error-correction of a finite-length LDPC code is for the Gallager-B decoder which was derived in [Chilappagari10] in relation to the girth g of the graph of the code, where the girth is the length of the shortest cycle present in the graph. It was shown that for a column-weight-three code of girth $g \geq 10$, the Gallager-B decoder could correct $(g/2-1)$ errors in $g/2$ iterations, and for girth $g=8$, the code required some additional constraints on the graph to ensure it can correct three errors. In this task, our main goal is propose new message-passing decoders that can be analyzed and proven to guarantee the correction of a greater number of errors than Gallager-B decoder in a finite number of iterations, while also requiring more relaxed conditions on the graph than Gallager-B so that codes with higher rates can be designed.

As a first step towards this goal, we proposed a new class of decoders called *decimation-enhanced finite alphabet iterative decoders (DFAIDs)* where the technique of *decimation* is incorporated into the framework of FAIDs as a tool to make the decoders more amenable to analysis while maintaining their good performance. Decimation, a method originally developed for solving constraint satisfaction problems in statistical physics, involves guessing the values of certain variables and fixing them to these values while continuing to estimate the remaining variables. We utilize decimation in a novel way by incorporating it into the decoding so that certain variable nodes are decimated after a few iterations of message passing, and a variable node is decimated based on its incoming messages at the end of some iteration. In this manner, decimation primarily serves as a guide to help the decoder to converge faster on low-weight error patterns. We first proposed a simple decimation scheme for a particular 3-bit precision 7-level FAID that is known to have good performance surpassing floating-point BP. Using this scheme as the basis, we propose a method to analyze the decoder and derive conditions on the graph of the code such that the decoder can be proven to correct a certain number of t errors in a finite number of iterations. As a test case, currently we are working on deriving conditions on the graph such that DFAIDs can correct up to t=4 errors in 3 iterations after decimation. Note that this would be a first result on guaranteed error-correction of a message-passing decoder other than Gallager-B decoder. Also note that for now, we only perform the analysis under a perfect decoder setting. Once we obtain the conditions on the graph to guarantee t=4 errors, we will utilize both the code satisfying these conditions and the DFAID as a decoder in the TK setting, and perform analysis in the presence of gate failures.

6. General Conclusion and Next Steps

Over the first year of i-RISC project our activities included analysis of Taylor-Kuznetsov memory architecture based on structural LDPC codes, under independent and data-dependent logic gate failures. We introduced a novel approach in gate failures modeling which enable us to capture data-dependence in more details, compared with state-of-the-art modeling. We especially investigated performance of one-step majority logic decoders in the presence of correlated gate failures. The analytical expression for BER of one-step majority logic decoders built from unreliable logic gates was derived.

In addition, a new class of bit flipping algorithms operating at perfect hardware was proposed. It was shown that introducing the collective error correction principle TBF algorithms can outperform min-sum and Gallager A/B message passing decoders.

We have also shown that based on the progressive constructing of Tanner graphs by avoiding trapping sets a code with good error-floor features can be constructed.

The topics presented in this deliverable will also be investigated in the second and the third year of the project. Our next steps include TK memory architecture performance derivation in the presence of hardware failures obtained by real measurements, development of analytical tools for memory analysis and attempts to design low complexity memory architectures. Based on promising result concerning TBF algorithms we will investigate the performance of memories constructed based on multi-bit bit flipping algorithms.

References

[**Ghosh10**] S. Ghosh, K. Roy, "Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era," *Proceedings of IEEE*, vol. 98, no. 10, pp. 1718-1751, Oct. 2010.

[**Hadjicostis05**] C. N. Hadjicostis and G. C. Verghese, "Coding approaches to fault tolerance in linear dynamic systems," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 210-228, Jan. 2005.

[**Varshney11**] L. Varshney, "Performance of LDPC codes under faulty iterative decoding," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4427–4444, July 2011.

[**Leduc-Primeau12**] F. Leduc-Primeau and W. Gross, "Faulty Gallager B decoding with optimal message repetition," in *Proc 50th Allerton Conf. Communication, Control, and Computing*, Oct. 2012, pp. 549–556.

[**Yazdi13**] S. M. SadeghTabatabaei Yazdi, H. Cho, L. Dolecek, "Gallager B decoder on noisy hardware," *IEEE Transactions on Communications*, vol. 61, no. 5, pp. 1660-1673, May 2013.

[**Huang13**] C. H. Huang and L. Dolecek, "Analysis of finite alphabet iterative decoders under processing errors," in *Proc IEEE Int. Conf. Acoustics, Speech, Sig. Proc.*, May 2013, pp. 5085-5089.

[**Ngassa13**] C. Kameni Ngassa, V. Savin and D. Declercq, "Min-Sum-based decoders running on noisy hardware", in *Proc. IEEE GLOBECOM*, Atlanta, USA, Dec. 2013.

[Taylor68-a] M. G. Taylor, "Reliable information storage in memories designed from unreliable components", *Bell System Technical Journal*, vol. 47, no. 10, pp. 2299-2337, 1968.

[Taylor68-b] M. G. Taylor, "Reliable computation in computing systems designed from unreliable components", *Bell System Technical Journal*, vol. 47, no. 10, pp. 2339-2366, 1968.

[**Kuznetsov73**] A. Kuznetsov, "Information Storage in a Memory Assembled from Unreliable Components", *Problems of Information Transmission*, vol. 9, no. 3 pp. 254-264, 1973.

[**Vasic07**] B. Vasic and S. K. Chilappagari, "An Information Theoretical Framework for Analysis and Design of Nanoscale Fault-Tolerant Memories Based on Low-Density Parity-Check Codes", *IEEE Transactions on Circuits and Systems I, Regular Papers*, vol. 54, no. 11, pp. 2438-2446, Nov. 2007.

[**Chilappagari07**] S. K. Chilappagari and B. Vasic, "Fault Tolerant Memories Based on Expander Graphs", In *Proceedings of IEEE Information Theory Workshop*, Tahoe City, CA, USA, pp. 126-131, Sep. 2007.

[**Chilappagari06-a**] S. K. Chilappagari, M. Ivkovic, and B. Vasic, "Analysis of One-step Majority Logic Decoders Constructed from Faulty Gates", In *Proceedings of IEEE Internationl Symposium on Information Theory*, Seattle, WA, USA, pp. 469-473, July 2006.

[**Chilappagari08**] S. K. Chilappagari, B. Vasic, and M. Marcellin, "Can the Storage Capacity of Memories Built from Unreliable Components Be Determined?", In *Proceedings of Information Theory and Applications Workshop*, San Diego, CA, USA, pp. 41-43, Jan. 2008.

[**Ivkovic06**] M. Ivkovic, S. K. Chilappagari, and B. Vasic, "Construction of Memory Circuits Using Unreliable Components Based on Low-Density Parity-Check Codes", In *Proceedings of IEEE Global Telecommunications Conference 2006*, *GLOBECOM '06*, San Fransisco, CA, USA, pp. 1-5, Nov. 2006.

[**Radhakrishnan07**] R. Radhakrishnan, S. Sankaranarayanan, and B. Vasic, "Analytical performance of one-step majority logic decoding of regular LDPC codes," in *Proc. 2007 IEEE Int. Symp. Inf. Theory,* June 2007. pp. 231–235.

[**Malandro13**] M. E. Malandro, "Integer compositions with part sizes not exceeding k," [Online Available] http://arxiv.org/abs/1108.0337

[MacKay99] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[**Shokrollahi02**] A. Shokrollahi, "An Introduction to Low-Density Parity-Check Codes," *Theoretical Aspects of Computer Science: Advanced Lectures*, Springer-Verlag New York, Inc., New York, NY, 2002.

[**Zyablov76**] V. Zyablov and M. Pinsker, "Estimation of the error-correction complexity for Gallager low-density codes," *Probl. Inf. Transm.*, vol. 11, no. 6, pp. 18–26, 1976.

[**Sipser96**] M. Sipser and D. Spielman, "Expander codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.

[**Burshtein08**] D. Burshtein, "On the error correction of regular LDPC codes using the flipping algorithm," *IEEE Trans. Inf. Theory*, vol. 54, no. 2, pp. 517–530, Feb. 2008.

[**Nouh04**] A. Nouh and A. Banihashemi, "Reliability-based schedule for bit-flipping decoding of low-density parity-check codes," *IEEE Trans. Commun.*, vol. 52, no. 12, pp. 2038–2040, Dec. 2004.

[**Ngatched09**] T. M. N. Ngatched, M. Bossert, A. Fahrner, and F. Takawira, "Two bit-flipping decoding algorithms for low-density parity-check codes," *IEEE Trans. Commun.*, vol. 57, no. 3, pp. 591–596, Mar. 2009.

[**Wu09**] X. Wu, C. Ling, M. Jiang, E. Xu, C. Zhao, and X. You, "New insights into weighted bit-flipping decoding," *IEEE Trans. Commun.*, vol. 57, no. 8, pp. 2177–2180, Aug. 2009.

[**Ngatched07**] T. M. N. Ngatched, F. Takawira, and M. Bossert, "A modified bit-flipping decoding algorithm for low-density parity-check codes," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2007, pp. 653–658.

[**Wadayama10**] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi, "Gradient descent bit flipping algorithms for decoding LDPC codes," *IEEE Trans. Commun.*, vol. 58, no. 6, pp. 1610–1614, Jun. 2010.

[**Miladinovic05**] N. Miladinovic and M. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 4, pp. 1594–1606, Apr. 2005.

[**Chan98**] A. Chan and F. Kschischang, "A simple taboo-based soft-decision decoding algorithm for expander codes," *IEEE Commun. Letters*, vol. 2, no. 7, pp. 183–185, Jul. 1998.

[**Nguyen13**] D. V. Nguyen, "Improving the error floor performance of LDPC codes with better codes and better decoders," 2013.

[**McKay10**] B. D. McKay, "Subgraphs of random graphs with specified degrees," in *Proc. Int. Congress of Mathematicians*, Hyderabad, India, Aug. 19–27 2010, pp. 2489–2501.

[**Tanner01**] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. 5th Int. Symp. Commun. Theory and Applications*, Ambleside, UK, Jul. 15–20 2001.

[**Nguyen12**] D. V. Nguyen, S. K. Chilappagari, B. Vasic, and M. W. Marcellin, "On the construction of structured LDPC codes free of small trapping sets," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2280–2302, Apr. 2012.

[**Vasic13-a**] "Two-bit bit flipping algorithms." [Online]. Available: http://www2.engr.arizona.edu/_vasiclab/ProjectsHome.php

[**Chilappagari10**] S. K. Chilappagari, D. V. Nguyen, B. Vasic, and M. W. Marcellin, "On trapping sets and guaranteed error correction capability of LDPC codes and GLDPC codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1600–1611, Apr. 2010.

[**Richardson03**] T. J. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conf. on Commun., Control and Computing*, 2003, pp. 1426–1435.

[**Vasic13-b**] B. Vasic, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology." [Online]. Available: http://www2.engr.arizona.edu/_vasiclab/project.php?id=9

[**Chilappagari06-b**] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasic, "Error floors of LDPC codes on the binary symmetric channel," in *Proc. IEEE Int. Conf. On Commun.*, vol. 3, Istanbul, Turkey, Jun. 2006, pp. 1089–1094.

[Ivkovic06-IT] 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, Aug. 2008.

[Planjery13] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders, Part 1: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4033–4045, Oct. 2013.

[Chilappagari10] S. K. Chilappagari, D. V. Nguyen, B. Vasic, and M. W. Marcellin, "Error correction capability of column-weight-three LDPC codes under the Gallager A algorithm—Part II," *IEEE Trans. Inf. Theory.*, vol. 56, no. 6, pp. 2626–2639, Jun. 2010.

Appendix A

Decimation-enhanced Finite Alphabet Iterative Decoders with Provable Guaranteed Error-Correction for LDPC Codes

Abstract: *Finite alphabet iterative decoders (FAID) were recently proposed for LDPC codes on the binary symmetric channel (BSC) which are capable of surpassing the belief propagation decoder in the error floor region , but with lower complexity and lower precision requirements than BP. The FAIDs were designed for column-weight-three codes with the goal of improving the error floor performance compared to BP on a given code using the knowledge of trapping sets. In this paper, we address the problem of analyzing the guaranteed error-correction capability of FAIDs for finite number of iterations by introducing decimation into the framework of FAIDs, giving rise to a new class of decoders called decimation-enhanced FAIDs. The technique of decimation, which is incorporated into the message update rule, involves fixing certain bits of the code to a particular value. Decimation is utilized in a novel manner during message-passing such that the decoder can be analyzed while maintaining the good performance of the original FAID. Further, decimation can reduce the number of iterations required to correct a fixed number of errors. We shall provide a simple decimation scheme for a particularly good 3-bit precision FAID for column-weight three codes on the BSC. Using this scheme, we shall describe a method to analyze the decoder for a finite number of iterations and derive conditions on the Tanner graph of code such that a fixed number of t errors are corrected in a finite number of I iterations. As a test case, we prove that under certain conditions satisfied by the Tanner graph of the code, the decoder can guarantee correction of $t = 4$ errors.*

A.1 Introduction

The design and analysis of message-passing (MP) algorithms for low-density parity-check (LDPC) [1] codes have received much attention over the last decade. Techniques such as density evolution [2] by Richardson and Urbanke, have been proposed for asymptotic anal-

ysis of MP decoders on LDPC code ensembles. For finite-length analysis of codes with a fixed number of iterations, methods such as the use of computation trees by Wiberg [3], pseudocodeword analysis by Kelly and Sridhara [4], and graph-cover decoding analysis by Vontobel and Koetter [5], have been proposed. The characterization of the error floor phenomenon of MP algorithms has also been well investigated using the notion of stopping sets for the binary erasure channel (BEC) [6] by Di et al., and using notions of trapping sets by Richardson [7] and instantons by Chernyak et al. [9] for other general channels. Burshtein and Miller proposed the technique of using expander arguments for MP for proving that code ensembles can correct a linear fraction of errors [10].

In spite of the aforementioned techniques proposed for finite-length analysis, the problem of analyzing a particular MP algorithm for a fixed number of iterations still remains a challenge, and the guaranteed error-correction capability of LDPC codes under MP decoding remains largely unknown. This is because the dynamics of MP becomes too complex beyond a certain number of iterations, and there is exponential growth in the number of nodes with the number of iterations in the computation trees of the decoders on the code. Although Burshtein and Miller's method of using expander arguments which allows for use of large number of iterations, provides bounds of great theoretical value, they are practically less significant. Moreover, for the binary symmetric channel (BSC), the problem of correcting a fixed number of errors assumes greater importance as it determines the slope of the error floor in the error-rate performance of the decoder [11]. Results on the guaranteed error-correction capability of LDPC codes is known only for the Gallager-B decoding which was derived by Chilappagari *et al.* [12] which passes only hard messages (1-bit precision messages), and which was derived for column-weight-three codes. They proved that for a column-weight-three code of girth $g \geq 10$, the Gallager-B decoder corrects $(g/2-1)$ errors in $g/2$ iterations, and for girth $g = 8$, the code required some additional constraints on the graph to ensure that it corrects $(g/2 - 1) = 3$ errors. However, no such results are known for any other MP decoder due to the prohibitive complexity of the analysis in the MP especially when the decoder utilizes soft messages. Therefore, it would be desirable to have an MP decoder that is able to correct a higher number of errors within the fewest possible iterations while allowing more relaxed conditions on the graph (instead of girth), and for which we will be able to provide performance guarantees in terms of error-correction capability. Even from a practical standpoint, this would be an attractive feature with many present-day applications requiring much higher decoding speeds and much lower target frame error rates.

Recently a new class of finite precision decoders, referred to as *finite alphabet iterative decoders* (FAID) [13] was proposed for LDPC codes on the BSC. For column-weight-three codes, it was shown that these decoders are capable of surpassing the belief propagation (BP) decoder in terms of error-rate performance as well as guaranteed error-correction, but at a much lower complexity and lower precision requirements to represent the messages than the BP decoder. These decoders were derived by identifying potentially harmful subgraphs that could be trapping sets present in any finite-length code and designing to correct error patterns on these subgraphs in an isolated manner. Although the numerical results in [13] demonstrated the efficacy of these decoders, providing provable statements in terms of guaranteed error correction capability still remains a difficult task since the convergence of the decoder for an error pattern in a trapping set is heavily influenced by the neighborhood of the trapping set in a non-trivial manner. This was also identified by Declercq *et al.* in [14],

where subgraphs induced by codewords were used in the decoder design.

In this paper, we address the problem of analyzing the FAIDs for finite number of iterations and guaranteed error-correction. This is achieved by introducing the technique of decimation into the framework of FAIDs thereby leading to a new class of decoders referred to as *decimation-enhanced finite alphabet iterative decoders*. Decimation, a method originally developed for solving constraint satisfaction problems in statistical physics, involves guessing the values of certain variables and fixing them to these values while continuing to estimate the remaining variables. In [15], Montanari *et al.* analyzed a BP-guided randomized decimation procedure that estimates the variables in the k-SAT problem. Dimakis and Wainwright used a similar notion in the form of facet guessing for linear programming (LP) based decoding in [16], and Chertkov proposed a bit-guessing algorithm in order to reduce error floors of LDPC codes under LP decoding [17].

In contrast, we utilize decimation in a novel way by incorporating it into the decoding of LDPC codes so that certain variable nodes are decimated after a few iterations of message passing, and a variable node is decimated based on its incoming messages at the end of some iteration. In this manner, decimation primarily serves as a guide to help the decoder to converge faster on low-weight error patterns. Our main insight is that the role of decimation should not necessarily be to correct errors, but to ensure that more variable nodes in the graph that initially receive right values from the channel are shielded from the erroneous messages emanating from the error nodes by decimating those correct variable nodes.

We first propose a simple decimation procedure for a particular 3-bit precision 7-level FAID algorithm (from [13]) that is known to have good error-rate performance on column-weight-three codes. Based on this scheme, we will provide a methodology to analyze the DFAID algorithm in order to prove the guaranteed correction of t errors in a finite number of decoding iterations I (where I is the number of MP iterations after the decimation procedure is completed). As a test case, we present the analysis for $t = 4$, and prove that on girth-8 column-weight-three codes whose Tanner graphs do not contain certain subgraphs (that will be described subsequently), the DFAID can correct $t = 4$ errors in $I = 3$ iterations after decimation. Note that this result on guaranteed error-correction is significant, since it is a first result presented for an MP decoder other than Gallager-B. Also note that the Gallager-B decoder fails to correct 4 errors on girth-8 column-weight-three codes, therefore showing that the proposed DFAID is a provably better decoder than Gallager-B decoder in terms of guaranteed error-correction.

The paper is organized as follows. Section A.2 provides the necessary preliminaries on LDPC codes and FAIDs. Section A.3 introduces the technique of decimation and provides the formal definitions on DFAIDs. Section A.4 presents the analysis of the DFAIDs for finite number of iterations with the test case of $t = 4$ and describes the methodology of proof. Finally Section A.5 presents some conclusions and final remarks describing the work in progress.

A.2 Preliminaries

A.2.1 LDPC codes and channel assumptions

Let G denote the Tanner graph of an (N, K) binary LDPC code $\mathcal{C}$ 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 $\mathcal{C}$ is said to have a regular column-weight d_v if all variable nodes in V have the same degree d_v. The set of neighbors of variable node v_i is denoted as $\mathcal{N}(v_i)$, and the set of neighbors of check node c_j is denoted by $\mathcal{N}(c_j)$. Let $\mathcal{N}(u)$ denote the set of neighbors of a node u in the graph G and let $\mathcal{N}(U)$ denote the set of neighbors of all $u \in U$. The girth of G is the length of shortest cycle present in G. Let $\mathcal{G}(N, M)$ denote a bipartite graph containing N variable nodes and M checks nodes. We shall use the notation

Let $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ denote a codeword of $\mathcal{C}$ 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 $\mathbf{r} = \{r_1, r_2, \ldots, r_N\}$. Let $\mathbf{e} = (e_1, e_2, \ldots, e_N)$ denote the *error pattern* introduced by the BSC with cross-over probability α such that $\mathbf{r} = \mathbf{x} \oplus \mathbf{e}$, and $\oplus$ is the component-wise modulo-two sum. The support of an error vector $\mathbf{e} = (e_1, e_2, \ldots, e_N)$, denoted by $\text{supp}(\mathbf{e})$, is defined as the set of all positions i such that $e_i \neq 0$. Let $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ 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 [2].

A.2.2 Finite Alphabet Iterative Decoders

An N_s-level FAID [13] denoted by $\mathcal{F}$, is a 4-tuple given by $\mathcal{F} = (\mathcal{M}, \mathcal{Y}, \Phi_v, \Phi_c)$. The messages are levels confined to a finite alphabet $\mathcal{M} = \{-L_s, \ldots, -L_2, -L_1, 0, L_1, L_2, \ldots, L_s\}$ consisting of $N_s = 2s + 1$ levels, where $L_i \in \mathbb{R}^+$ and $L_i > L_j$ for any $i > j$. The sign of a message $x \in \mathcal{M}$ can be interpreted as the estimate of the bit (positive for zero and negative for one) associated with the variable node to or from which x is being passed, and the magnitude $|x|$ as a measure of how reliable this value is. The message 0 in the alphabet can be interpreted as an erasure message.

The set $\mathcal{Y}$ denotes the set of possible channel values. For the case of BSC, $\mathcal{Y} = \{\pm \text{C}\}$, where $\text{C} \in \mathbb{R}^+$. By convention, we use the mapping $0 \to \text{C}$ and $1 \to -\text{C}$. Let $m_1, m_2, \ldots, m_{l-1}$ denote the $l - 1$ extrinsic incoming messages of a node (check or variable) of degree l which are used in the calculation of the outgoing message.

The function $\Phi_c : \mathcal{M}^{d_c - 1} \to \mathcal{M}$ is used for update at a check node with degree d_c and is defined as

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

The function $\Phi_v : \mathcal{Y} \times \mathcal{M}^{d_v - 1} \to \mathcal{M}$ is a map used for update at a variable node with degree d_v. Note that since the update function Φ_c is the same in all FAIDs considered, an N_s-level FAID is uniquely defined by the function Φ_v.

The function Φ_v can described as a closed form function or simply as a d_{v-1}-dimensional array or look-up table (LUT). In this paper, we shall only use the LUT form. For $d_v = 3$, a 2D array or LUT is sufficient to describe the map Φ_v for a channel value -C, and the map for channel value +C can be deduced by the property of symmetry. An example of a 7-level FAID described in LUT form is provided in Table A.1.

Table A.1: Φ_v of a 7-level FAID for a node v_i with $d_v = 3$ and $y_i = +\mathrm{C}$

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

Note that the maps defining Φ_v must satisfy the following properties.

Property 1 (Property of symmetry) $\Phi_v(y_i, m_1, ..., m_{d_v-1}) = -\Phi_v(-y_i, -m_1, ..., -m_{d_v-1})$

Property 2 (Property of monotonicity) $\Phi_v(y_i, m_1, \ldots, m_{d_v-1}) \geq \Phi_v(y_i, m'_1, \ldots, m'_{d_v-1})$ *when* $m_j \geq m'_j \ \forall \ j \in \{1, \ldots, d_v - 1\}$.

A.2.3 Trapping sets and stopping sets

We begin by providing the definition of a trapping set as originally defined by Richardson in [7].

Definition 1 *Given a decoder input* $\mathbf{y}$*, a trapping set (TS) for an iterative decoder denoted by* $\mathbf{T}(\mathbf{y})$ *is a non-empty set of variable nodes in* G *that are not corrected at the end of a given number of iterations.*

A common notation used to denote a TS is (a, b), where $a = |\mathbf{T}|$, and b is the number of odd-degree check nodes in the subgraph induced by $\mathbf{T}$.

Let $\mathcal{T}(a, b)$ 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 $\mathcal{T}$ if there exists a subset of variable nodes $\mathbf{T}$ in G whose induced subgraph is isomorphic to $\mathcal{T}(a, b)$. A TS is said to be elementary if $\mathcal{T}$ 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 [7, 8]. Henceforth, for convenience, whenever we refer to a TS, we will implicitly refer to its topological structure $\mathcal{T}$.

Stopping sets [6] are a sub-class of trapping sets that were introduced to characterize failures on the BEC. Note that for the BEC, a transmitted bit is either received correctly or is erased. Let S denote a subset of variable nodes and let $\mathcal{N}(S)$ denote the set of neighbors $\{\mathcal{N}(v_i) : \forall v_i \in S\}$.

Definition 2 *A stopping set is a subset of variable nodes S in the graph G, such that every neighbor in $\mathcal{N}(S)$ is connected to subset S at least twice [6].*

A.2.4 Isolation assumption

The isolation assumption is a pivotal notion that enables us to analyze the message passing of a particular decoder on potentially harmful subgraphs (or trapping sets). It is a specific condition on the neighborhood of the subgraph such that the messages flowing into the subgraph from its neighborhood are not in any way influenced by the messages flowing out of the subgraph for certain number of iterations. Before we formally define the concept of isolation assumption, we first require the notion of computation tree [18] and introduce some notations. Note that during the analysis of any decoders, the all-zero codeword assumption is used.

Definition 3 *A computation tree corresponding to a message-passing decoder of the Tanner graph G is a tree that is constructed by choosing an arbitrary variable node in G as its root and then recursively adding edges and nodes to the tree that correspond to the messages passed during decoding on G up to a certain number of iterations. For each vertex that is added to the tree, the corresponding node update function in G is also copied.*

Let $\mathscr{T}_i^k(G)$ be the computation tree of graph G corresponding to a decoder $\mathscr{F}$ enumerated for k iterations with variable node $v_i \in V$ as its root.

Let H be the induced subgraph of a trapping set (a, b) contained in G with variable node set $P \subseteq V$ and check node set $W \subseteq C$. Let $W' \subseteq W$ denote the set of degree-one check nodes in the subgraph H. Let $P' \subseteq P$ denote the set of variable nodes in H where each variable node has at least one neighbor in W'. During decoding on G, for a node $v_i \in P'$, let μ_l denote the message that v_i receives from its neighboring degree-one check node in H in the l^{th} iteration.

Definition 4 *A vertex $w \in \mathscr{T}_i^k(G)$ is said to be a descendant of a vertex $u \in \mathscr{T}_i^k(G)$ if there exists a path starting from vertex w to the root v_i that traverses through vertex u. The set of all descendants of the vertex u in $\mathscr{T}_i^k(G)$ is denoted as $\mathcal{D}(u)$. For a given vertex set U, $\mathcal{D}(U)$ (with some abuse of notation) denotes the set of descendants of all $u \in U$.*

Definition 5 *A vertex $w \in \mathscr{T}_i^k(G)$ is said to be a child of a vertex $u \in \mathscr{T}_i^k(G)$ if w is directly connected to u along the path traversing to the root. A vertex that does not have any child nodes in $\mathscr{T}_i^k(G)$ is called a leaf node.*

Note that for the computation tree $\mathscr{T}_i^k(G)$, all its leaf variable nodes lie at a depth of $2k$ from the root of the tree. The variable nodes in the tree that lie at a depth of $2j$ from the root are said to be at the j^{th} level of the tree.

Definition 6 *$\mathscr{T}_i^k(H)$ is called the computation tree of the subgraph H enumerated for k iterations for the decoder $\mathscr{F}$, if $\forall\ c_j \in W'$, μ_l is given for all $l \leq k$, and if the root node $v_i \in P$ requires only the messages computed by the nodes in H and μ_l to compute its binary hard-decision value.*

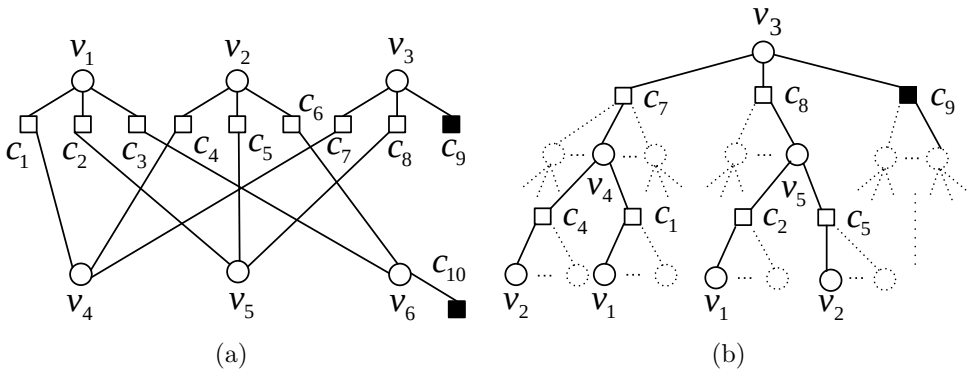


Figure A.1: Subgraph H corresponding to a $\mathcal{T}(6,2)$ trapping set contained in G: (a) Tanner graph of H; (b) computational tree $\mathcal{T}_3^2(G)$

We now provide the definition for isolation assumption as follows.

Definition 7 (Isolation assumption) *Let H be a subgraph of G induced by $P \subseteq V$ with check node set $W \subseteq C$. The computation tree $\mathscr{T}_i^k(G)$ with the root $v_i \in P$ is said to be isolated if and only if for any node $u \notin P \cup W$ in $\mathscr{T}_i^k(G)$, u does not have any descendant belonging to $P \cup W$. If $\mathscr{T}_i^k(G)$ is isolated $\forall v_i \in P$, then the subgraph H is said to satisfy the isolation assumption in G for k iterations.*

Essentially the isolation assumption of a subgraph validates the number of iterations for which message passing can be carried out on a subgraph in an isolated manner ignoring its neighborhood, which is done by treating it as if it were an arbitrary Tanner graph (of some code), provided that the messages passed from the degree-one check nodes of H in each iteration are known or determined a priori.

Note that the isolation assumption is a purely topological condition on the neighborhood of H, as it is dependent only on the particular graph G containing H and not on the decoder being used. Also note that the isolation assumption can still be satisfied even when there are nodes in H that appear multiple times in $\mathscr{T}_i^k(G)$. Whereas Gallager's independence assumption [1] will be violated if any node in H is repeated in $\mathscr{T}_i^k(G)$. Hence, isolation assumption is a weaker condition than independence. For clarity, we illustrate with an example shown in Fig. A.1.

Example 1 *Let us assume that the graph G of code $\mathcal{C}$ contains a subgraph H corresponding to a $\mathcal{T}(6,2)$ trapping set. Fig. A.1 shows the subgraph H, and the computation tree $\mathcal{T}_3^2(G)$ of graph G with v_3 as its root enumerated for two iterations. The ■ denotes a odd-degree check node present in H. The solid lines represent connections within subgraph H and the dotted lines represent connections from the rest of the graph G outside the subgraph H. The isolation assumption is satisfied for two iterations if none of the descendants of a node u which is outside H and represented by a dotted circle in $\mathcal{T}_3^2(G)$ are nodes in H. But the independence assumption does not hold for two iterations.*

In order to enable the message passing of a particular decoder on a subgraph H under isolation assumption, we need to know the messages passed from the degree-one check nodes of H in each iteration. Assuming that the decoder $\mathscr{F}$ is a FAID, this can be computed without knowledge of its neighborhood through the following theorem (under all-zero codeword assumption).

Theorem 1 (Isolation theorem) *Let H be a subgraph of a 3-left-regular graph G induced by $P \subseteq V$ with check node set $W \subseteq C$. Let $W' \subseteq W$ denote the set of degree-one check nodes in H and let $P' \subseteq P$ denote the set of variable nodes in H for which each has at least one neighbor in W'. Consider decoding on G with FAID $\mathscr{F}$. If $\mathbf{r}$ is received from the BSC such that $\mathrm{supp}(\mathbf{r}) \subseteq P$, and if H satisfies the isolation assumption in G for k iterations, then for each $c_j \in W'$, the message from c_j to its neighbor in H in the l^{th} iteration denoted by μ_l, is the output of $\Phi_v(\mathrm{C}, \mu_{l-1}, \mu_{l-1})$ $\forall l \leq k$, with $\mu_0 = 0$.*

Proof: Firstly, since $\mathrm{supp}(\mathbf{r}) \subseteq P$ and all messages are initialized to zero, all nodes $v_i \in V \backslash P$ are initially correct, and every variable node initially sends $\Phi_v(y_i, 0, 0)$.

Now consider a computation tree $\mathscr{T}_s^l(G)$ where $l \leq k$ with $v_s \in P'$ as its root. Due to the isolation assumption of H in G satisfied for k iterations, for any $c_j \in \mathcal{N}(v_s) \cap W'$, $\mathcal{D}(c_j) \cap (P \cup W) = \emptyset$. Therefore all nodes $v_i \in \mathcal{D}(c_j)$ are initially correct. Let q denote the level of depth involving variable nodes in $\mathscr{T}_s^l(G)$ such that level $q = 0$ denotes the base and level $q = l$ denotes the root of the tree. Let $\mathcal{D}^{(q)}(c_j)$ denote the variable node descendants of $c_j \in \mathcal{N}(v_s) \cap W'$ at level q of the tree $\mathscr{T}_s^l(G)$. At $q = 1$, due to the nature of Φ_c for FAID $\mathscr{F}$ along with the fact that every correct node initially sends $\Phi_v(\mathrm{C}, 0, 0)$, any $v_i \in \mathcal{D}^{(1)}(c_j)$ receives the message $\mu_1 = \Phi_v(\mathrm{C}, 0, 0)$, from its leaf check nodes. Again, since $\mathcal{D}(c_j) \cap (P \cup W) = \emptyset$, at $q = 2$, any $v_i \in \mathcal{D}^{(2)}(c_j)$ receives $\mu_2 = \Phi_v(\mathrm{C}, \mu_1, \mu_1)$. By induction, this can be generalized to any level q, where any node $v_i \in \mathcal{D}^{(q)}(c_j)$, receives $\mu_q = \Phi_v(\mathrm{C}, \mu_{q-1}, \mu_{q-1})$ from its leaf check nodes. Therefore the root v_s receives $\mu_l = \Phi_v(\mathrm{C}, \mu_{l-1}, \mu_{l-1})$. ■

Although the theorem is stated specifically for column-weight-three codes, it can be generalized to left-regular codes of higher column-weight as well. Also the isolation theorem can be restated for the min-sum decoder through the following corollary.

Corollary 1 *Consider the min-sum decoder for column-weight-three codes with $\mathcal{Y} = \{\pm 1\}$. If subgraph H contained in G satisfies the isolation assumption for k iterations, and if all variable nodes outside H are initially correct, then μ_l of the degree-one check node for the min-sum decoder is $2\mu_{l-1} + 1$.*

Corollary 2 *If H is a subgraph contained in G such that it satisfies the isolation assumption for k iterations, and if all variable nodes outside H are initially correct, then the computation tree $\mathscr{T}_i^k(G)$ with $v_i \in P$ is equivalent to $\mathscr{T}_i^k(H)$, provided μ_l for each degree-one check node in H is computed using the isolation theorem.*

A.3 Decimation: A tool for analysis

A.3.1 A motivating example

We illustrate the benefit of decimation through the following example. Note once again that we shall use the all-zero codeword assumption for the analysis of decoders.

Let $\mathcal{N}(u)$ denote the set of neighbors of a node u in the graph G and let $\mathcal{N}(U)$ denote the set of neighbors of all $u \in U$. Let $m^{(k)}(v_i, c_j)$ denote the message being passed from a variable node v_i to the check node c_j, in the k^{th} iteration, and let $m^{(k)}(c_j, v_i)$ be defined similarly. Let $m^{(k)}(v_i, \mathcal{N}(v_i))$ denote the set of outgoing messages from v_i to all its neighbors in the k^{th} iteration, and let $m^{(k)}(c_j, \mathcal{N}(c_j))$ be defined similarly. Recall that $\hat{x}_i^{(k)}$ denotes the bit value of a variable node $v_i \in V$ decided by the decoder at the end of the k^{th} iteration.

Consider a particular 4-error pattern on a Tanner graph G, such that the induced subgraph H of the four nodes in error is an 8-cycle as shown in Fig. A.2. In the figure, ● represents a variable node initially in error, and ○ represents an initially correct node that is in the neighborhood outside H. The ■ and □ denote the degree one and degree two check nodes in the induced subgraph respectively. Let $V'=\{v_1, v_2, v_3, v_4\}$ denote the set of nodes initially in error in H. Let $C^1=\{c_1, c_3, c_5, c_7\}$ denote the set of degree one checks and $C^2=\{c_2, c_4, c_6, c_8\}$ denote the set of degree two checks in H.

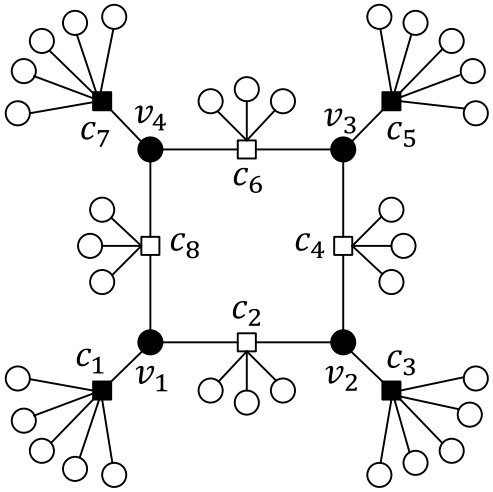


Figure A.2: Subgraph induced by the 4-error pattern which forms an 8-cycle

We shall now examine the behavior of MP on this particular error configuration from the context of N_s-level FAIDs without any assumptions on its neighborhood. Messages with a positive sign will be referred to as *good* messages, and messages with a negative sign will be referred to as *bad* messages. Also a message is referred to as *strongly* good (bad) or *weakly* good (bad) based on the magnitude of the message. For instance, a weakly good or bad message refers to $\pm L_1$, and a strongly good or bad message refers to $\pm L_i$ where $L_i > L_1$.

Assuming that $\Phi_v(\mathrm{C}, 0, 0) = L_1$, in the first iteration, for all $v_i \in V'$, $m^{(1)}(v_i, \mathcal{N}(v_i))$ will consist of weakly bad messages, and for all $v_j \in \mathcal{N}(C^1 \cup C^2) \backslash V'$, $m^{(1)}(v_j, \mathcal{N}(v_j) \cap (C^1 \cup C^2))$ entering into the subgraph will consist of weakly good messages. In the second iteration, for all $v_i \in V'$, $m^{(2)}(v_i, \mathcal{N}(v_i) \cap C^2)$ will consist of either weakly good or weakly bad messages depending on the Φ_v, but $m^{(2)}(v_i, \mathcal{N}(v_i) \cap C^1)$ which consists of messages sent to check nodes in C^1, will be strongly bad assuming $\Phi_v(-\mathrm{C}, -L_1, -L_1) = -L_2$. As a result, variable nodes $v_i \in \mathcal{N}(C^1) \backslash V'$ will start receiving a strongly bad message on at least one of its

edges. Now if the decoder does not converge within the next few iterations, then these bad messages become more strongly bad with circulation of bad messages within H, and this can subsequently spread further to other nodes in the graph outside H depending how dense the neighborhood is. Eventually too many nodes get corrupted by the bad messages being propagated in the graph causing the decoder to fail.

There are some important observations to note from the above discussion. Firstly, at the k^{th} iteration, there may have been many variable nodes $v_i \notin \mathcal{N}(C^1 \cup C^2)$ whose decided bit value $\hat{x}_i^{(k')}$ converged to the right value in some $k' < k$ iteration, but eventually these nodes became corrupted in the k^{th} iteration due to the bad messages flowing out of the subgraph. Secondly, if certain variable nodes initially correct in the neighborhood of H, are isolated from the bad messages possibly through decimation, then the decoder is more likely to converge. This is the case especially when there are relatively fewer errors introduced by the channel that are localized to a particular part of the graph (such as the 4-cycle), and this typically occurs in the high SNR (error floor) region. This is precisely where decimation becomes beneficial and important.

From the example, we can see that the role of decimation is inherently linked to the concept of isolation assumption which was defined in the previous section. Recall that the isolation assumption is a condition on the neighborhood of a subgraph contained in the graph such that the messages passed within the subgraph as well as messages entering into it from outside are not influenced by the messages being passed in its neighborhood and vice versa for certain number of iterations. In this context, the role of decimation can be understood as trying to isolate the subgraph induced by the nodes in error from the rest of the graph, so that the rest of the graph converges quickly and in turn helps to correct the nodes initially in error.

It is important to note once again that the emphasis with regards to using decimation is on low-weight error patterns typically associated with harmful trapping sets and being able to correct these patterns in the fewest possible iterations. An error pattern is considered to be low-weight if its weight is less than or equal to $\lfloor (d_{min} - 1)/2 \rfloor$.

A.3.2 Decimation-enhanced FAIDs

We now formally define the concept of decimation, introduce the required notations, and describe the novel way in which it is incorporated into the message passing of FAIDs. Note that the definitions are general enough to be applicable to any column-weight-d_v code, but we will eventually restrict our discussion to only column-weight-three codes.

Definition 8 *A variable node v_i is said to be decimated at the end of l^{th} iteration if $\hat{x}_i^{(k)}$ is set to $\hat{x}_i^*$ $\forall k > l$. Then $m^{(k)}(v_i, \mathcal{N}(v_i)) = \{(-1)^{\hat{x}_i^*} L_s\}$, $\forall k \geq l$ irrespective of its incoming messages $m^{(k)}(\mathcal{N}(v_i), v_i)$, i.e., v_i will always send the strongest possible messages.*

Note that if a node v_i is decimated at the end of l^{th} iteration, then this is equivalent to effectively deleting all its descendants $\mathcal{D}(v_i)$ in the computation tree $\mathscr{T}_i^k(G)$ $\forall k > l$ since the node always sends $(-1)^{b_i^*} L_s$ to its parent.

A *decimation rule* $\beta : \mathcal{Y} \times \mathcal{M}^{d_v} \rightarrow \{-1, 0, 1\}$ is a function used at the end of some l^{th} iteration by the decoder to decide whether a variable node should be decimated and what

value it should be decimated to based on the incoming messages and the channel value in the l^{th} iteration. Let γ_i denote the output of a decimation rule applied to a node v_i. If $\gamma_i = 0$, then the node is not decimated. If $\gamma_i = 1$, then the node is decimated with $\hat{x}_i^* = 0$, and if $\gamma_i = -1$, then the node is decimated with $\hat{x}_i^* = 1$.

There are two key aspects to note regarding the application of a decimation rule that add to the novelty of how we incorporate decimation into FAIDs.

1) The decimation rule is applied after messages are passed iteratively for some l iterations.

2) After each instance of applying the decimation rule, all messages are cleared to zero (which is practically restarting the decoder except that the decimated nodes remain decimated).

The first aspect implies that the decoder itself decides which nodes to decimate after passing messages for some number of iterations, which is in contrast to existing approaches that use decimation. The second aspect aids in preventing the growth of bad messages in the graph as well as in simplifying the analysis. More on the rationale behind this will be discussed later. We shall refer to each instance of applying a decimation rule on all variable nodes as a *decimation round*.

For now we only consider the use of a single decimation rule β which may be used in different iterations. We will later generalize to using muliple decimation rules when we discuss adaptive decimation. We now formally introduce the class of decimation-enhanced N_s-level FAIDs for the BSC.

A decimation-enhanced N_s-level FAID (DFAID) denoted by $\mathscr{F}^D$ is defined as a 4-tuple given by $\mathscr{F}^D = (\mathcal{M}, \mathcal{Y}, \Phi_v^D, \Phi_c)$, where $\mathcal{M} = \{-L_s, \ldots, -L_1, 0, L_1, \ldots, L_s\}$, $\mathcal{Y} = \{\pm \mathrm{C}\}$, and Φ_c are the same as defined for a N_s-level FAID. The map $\Phi_v^D : \mathcal{Y} \times \mathcal{M}^{d_v - 1} \times \{-1, 0, 1\} \to \mathcal{M}$ is similar to Φ_v of the FAID $\mathscr{F}$ with the minor difference that it uses the output of β in some l^{th} iteration as an additional argument in the function. Let m_1 and m_2 denote the extrinsic incoming messages to a node v_i with $d_v = 3$. Then Φ_v^D is defined as

$$\Phi_v^D(y_i, m_1, m_2, \gamma_i) = \begin{cases} \Phi_v(y_i, m_1, m_2), & \gamma_i = 0 \\ \gamma_i L_s, & \gamma_i = \pm 1 \end{cases}$$

In the proposed framework of DFAIDs, a decimation rule β must satisfy certain important properties. For ease of exposition, the properties are specified below for a degree-3 variable node.

1. $\beta(\mathrm{C}, m_1, m_2, m_3) = -\beta(-\mathrm{C}, -m_1, -m_2, -m_3)$
   $\forall m_1, m_2, m_3 \in \mathcal{M}$

2. $\beta(\mathrm{C}, m_1, m_2, m_3) \neq -1$ and $\beta(-\mathrm{C}, m_1, m_2, m_3) \neq 1$ $\forall m_1, m_2, m_3 \in \mathcal{M}$

3. Given $m_1, m_2, m_3 \in \mathcal{M}$, if $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$, then $\beta(\mathrm{C}, m_1', m_2', m_3') = 1$ $\forall m_1', m_2', m_3' \in \mathcal{M}$ such that $m_1' \geq m_1$, $m_2' \geq m_2$, and $m_3' \geq m_3$.

Property 1 just enforces symmetry on the function β.

Property 2 implies that a node v_i can be decimated to zero only if $y_i = \mathrm{C}$ and to one only if $y_i = -\mathrm{C}$, meaning a node can be decimated only to its received value r_i from BSC. An important consequence of this is that, a node initially correct will never be decimated to a wrong value and a node initially wrong will never be decimated to the correct value. This means that, by Property 2, decimation will not by itself correct nodes initially in error, but rather acts as a reinforcer to the nodes initially correct by preventing them from being corrupted when they are decimated. The rationale behind this is that since we are focusing on low-weight error pattens, the number of nodes initially in error are significantly small compared nodes initially correct, so we rely on the nodes initially correct to drive the decoder to converge. Moreover, this simplifies the analysis of the decoders. It is evident then that a necessary condition for successful decoding is that no node initially in error is decimated.

Property 3 ensures that there is monotonicity in the inputs with relation to the output of the decimation rule, and this is required due to the nature of decimation. For instance, if a degree-3 variable node initially correct is decimated when its incoming messages are L_1, L_1, and L_2, then it must be decimated when it receives L_2, L_2, and L_2.

For convenience, we shall refer to variable nodes initially in error in G as *error nodes* and variable nodes initially correct in G as *correct nodes* throughout this paper. We now propose a simple decimation scheme on column-weight-three codes with the underlying FAID being a 7-level FAID.

A.3.3 Proposed scheme for 7-level FAID on column-weight-three codes

Let N_d denote the number of decimation rounds carried out by the decoder with a given decimation rule β. The decimation scheme essentially specifies exactly which iteration the decimation rule is applied, and the number of such decimation rounds. For the scheme, we only consider 7-level FAIDs whose variable node update maps Φ_v satisfy $\Phi_v(\mathrm{C}, 0, 0) = L_1$, $\Phi_v(\mathrm{C}, L_1, L_1) = L_2$, and $\Phi_v(\mathrm{C}, L_2, L_2) = L_3$. This is because, all the particularly good 7-level FAIDs were found to have this property.

The scheme we propose involves performing the first round of decimation at the end of the third iteration, then restarting the decoder and allowing one iteration of message passing after which the next decimation round is performed, and this is repeated until a total of N_d decimation rounds have been completed. Note that once a node is decimated in a particular decimation round, it remains decimated for the remainder of the entire decoding, and the decimation rule is applied only on the non-decimated nodes. The skeleton of the proposed scheme for a 7-level FAID on a column-weight-three code is given in Algorithm A.3.3.

There are two main reasons for performing the first decimation round at the end of the third iteration (which are specific to the 7-level FAID). Firstly, since the bit value of a node remains fixed once the node is decimated, we want to ensure that nodes being decimated have reliably converged, which is partially determined by the strength of messages entering the node. Therefore, message passing must be done for enough iterations to allow the possibility of a message with the highest possible strength being passed in the graph. For the 7-level

Algorithm 1 Decimation-enhanced FAID algorithm

1) Initialize $\gamma_i = 0 \ \forall v_i \in V$.

2) Run the decoder for three iterations using update maps Φ_v and Φ_c defined for the 7-level FAID.

3) Perform decimation using the rule β for every $v_i \in V$, which constitutes the first decimation round.

4) Restart the decoder by resetting all the messages to zero and pass messages for one iteration. This implies that a decimated node v_i will send $\gamma_i L_3$ and a non-decimated nodes v_j will send $\Phi_v(y_j, 0, 0)$.

5) Repeat step 3) only for nodes $v_i \in V$ whose $\gamma_i = 0$, followed by 4) until N_d decimation rounds have been completed.

6) Run the decoder for the remainder of iterations using maps Φ_v^D and Φ_c.

FAIDs considered, since $\Phi_v(\mathrm{C}, 0, 0) = L_1$, and $\Phi_v(\mathrm{C}, L_1, L_1) = L_2$, at least three iterations are required for an L_3 or $-L_3$ to be passed by a variable node. Therefore we need to allow at least three iterations of message passing before performing the decimation.

Secondly, as illustrated in the example discussed previously, it would be desirable to decimate the correct nodes as much as possible before they get influenced by the bad messages emanating from the error nodes. Therefore, the decoder must not carry out too many iterations of message passing before performing the first decimation. Based on the two reasons, the choice of allowing three iterations for message passing seems to be an appropriate choice. Moreover, decimating nodes after only three iterations makes the algorithm much more amenable to analysis. For instance, it becomes possible now to analyze under what conditions of the graph will an error node get decimated. We shall in fact derive such conditions for the previous example of the 4-cycle.

With regards to restarting the decoder after each decimation round, the rationale behind this is to allow the decimated correct nodes to drive the decoder convergence in the right direction by preventing the growth of bad messages in the graph (as all messages are reset to zero), assuming that no error node is decimated in the first decimation round. Furthermore, by performing subsequent decimation rounds at the end of just one iteration after restarting the decoder, it leads to the following important lemma.

Lemma 1 *Consider decoding on G with DFAID $\mathscr{F}^D$ where the first decimation round is performed at the end of third iteration, and subsequent decimation rounds are performed after resetting all messages to zero and passing messages for one iteration. Let β be such that $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$. Given an error pattern on G, if no error node gets decimated in the first decimation round, then no error node will get decimated in the subsequent decimation rounds.*

Proof: Since no error node is decimated in the first decimation round, an error node will send $\Phi_v(-\mathrm{C}, 0, 0) = -L_1$ in the first iteration after resetting all the messages to zero. In the

Table A.2: Set Ξ consisting of all message triples such that $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$

m_1	m_2	m_3
L_3	L_3	L_3
L_3	L_3	L_2
L_3	L_3	L_1
L_3	L_3	0
L_3	L_3	-L_1

m_1	m_2	m_3
L_3	L_2	L_2
L_3	L_2	L_1
L_3	L_2	0
L_3	L_2	-L_1
L_3	L_1	L_1

m_1	m_2	m_3
L_3	L_1	0
L_3	L_1	-L_1
L_3	0	0
L_2	L_2	L_2
L_2	L_2	L_1

worst-case scenario for the error node, each of its neighboring checks can be connected to an additional error node, in which case it receives a $-L_1$ from all three of its neighbors at the end of one iteration. But since $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$ implying that $\beta(-\mathrm{C}, -L_1, -L_1, -L_1) = 0$ by Property 1, an error node will not get decimated. This holds for any subsequent decimation round. ■

As a result of Lemma 1, if β is defined so that $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$, then we need to only ensure that an error node is not decimated at the end of third iteration, which simplifies analysis. Moreover, any number of decimation rounds can be used without being concerned about an error node being decimated. We now provide some insights into the design of the decimation rule β.

A.3.4 Design of decimation rule β

It is clear from the previous discussion that the design of β is critical for ensuring that none of the error nodes are decimated on low-weight error patterns while more correct nodes are decimated. Also due to Lemma 1, we only consider decimation rules that satisfy $\beta(\mathrm{C}, L_1, L_1, L_1) = 0$.

In order to define the decimation rule β, we define a set Ξ that consists of all unordered triples $(m_1, m_2, m_3) \in \mathcal{M}^3$ such that $\beta(\mathrm{C}, m_1, m_2, m_3) = 1$. Note that for any unordered triple $(m_1, m_2, m_3) \in \Xi$, $\beta(-\mathrm{C}, -m_1, -m_2, -m_3) = -1$ by property 1, so Ξ is sufficient to completely specify β.

The design of the rule β can now be considered as a procedure of selecting the unordered triples to be included in the set Ξ. This depends not only on the particular graph G of a code but also on the particular underlying FAID being used. Given an underlying 7-level FAID, we would like to do the selection with particular focus on correcting small number of errors typically associated with trapping sets in the error floor region. Referring back to the example of the 8-cycle shown in Fig. A.2, a good decimation rule would be one where γ_j for most or all nodes $v_j \in \mathcal{N}(C^1 \cup C^2) \backslash V'$ is 1 and γ_i for nodes $v_i \in V'$ is 0 at the end of all decimation rounds. Let us assume we are designing a decimation rule β for a particularly good 7-level FAID identified in the previous chapter, whose Φ_v is defined by Table A.1. Although we do not describe a rigorous method to design the rule β, we highlight three main points that are important to consider for its design.

Firstly, when considering whether a particular unordered triple (m_1, m_2, m_3) should be included in Ξ or not, the number of positive messages and negative messages in the triple as well as their magnitudes play a role in the selection. For instance, (L_3, L_1, L_1) may be a more

plausible candidate than $(L_3, L_2, -L_2)$, because the former has three positive messages, and also the latter has a high negative message in $-L_2$.

Secondly, the inherent nature of the particular Φ_v used in the FAID must be taken into consideration during selection. For this, we need to look at what outgoing messages a variable node would send when a particular triple is the set of incoming messages, and then decide if this is good for selection. For instance, given the 7-level FAID defined by Table A.1, the triple (L_2, L_2, L_1) might be a more plausible candidate than $(L_3, L_2, -L_2)$ since the outgoing messages for the former would be all L_3, but the outgoing messages for the latter would be L_3, L_3, and L_1.

Thirdly, if the harmful subgraphs present in graph G of a given code are known, the isolation assumption can be a useful tool to identify which triples should be avoided for inclusion in Ξ. If our goal is to correct certain number of errors , say t, the MP with the 7-level FAID would be analyzed for a given t-error pattern on the subgraph under isolation assumption. If a particular triple matches with the messages received by an error node in the subgraph at the end of third iteration, this triple must be not be included in Ξ.

Using the approach outlined in the above three points, we provide a decimation rule designed, with the underlying 7-level FAID defined by Table A.1, for the well-known $(155, 64)$ Tanner code. The rule is defined by Table A.2 which shows all unordered triples included in the set Ξ. The rule was designed with the goal of reducing the number of iterations required to guarantee a correction of 5 errors on the code.

Let us now revert back to our motivating example which involves an error pattern whose induced subgraph is an 8-cycle. For the particular 7-level DFAID, we can now analyze whether or not any error nodes get decimated assuming that their neighborhood satisfies certain topological conditions. This is illustrated through the following theorem. Note that the 7-level DFAID used in the theorem comprises of the decimation rule β defined by Table A.2, and the Φ_v defined by Table A.1.

Theorem 2 *Consider the decoding of a 4-error pattern by the 7-level DFAID on graph G whose support is contained in an 8-cycle. If G has girth-8, and no three check nodes of the 8-cycle share a common variable node, then the error nodes will not get decimated in any decimation round.*

Proof: Firstly note that by virtue of Φ_v of the 7-level FAID (Table A.1), the highest magnitude of a message that any node $v_i \in V$ can send is L_1 in the first iteration and L_2 in the second iteration. Since a node $v_j \in \mathcal{N}(C^1 \cup C^2) \backslash V'$ can be connected to at most two checks in subgraph, the node v_j in the worst case recieves two $-L_1$ messages from checks in $C^1 \cup C^2$ and L_1 from outside at the end of first iteration. Node $v_i \in V'$ will also receive two $-L_1$ messages from check nodes in C^2 and L_1 from $c_k \in C^1 \cap \mathcal{N}(v_i)$. At the end of the second iteration, the node $v_i \in V'$ will once again receive two $-L_1$ from checks in C^2, and L_1 from $c_k \in C^1$. This means that node v_i will receive two $-L_1$ messages once gain from checks in C^2 at the end of third iteration. In order for it to be decimated, from Table A.2, it must receive $-L_3$ from $c_k \in C^1 \cap \mathcal{N}(v_i)$. This means that the node v_j in the worst case has to receive at least one $-L_3$ at the end of the second iteration, but this is not possible by virtue of Φ_v in the second iteration. Hence, a node initially in error can not get decimated at the end of third iteration and using Lemma 2, will never get decimated. ■

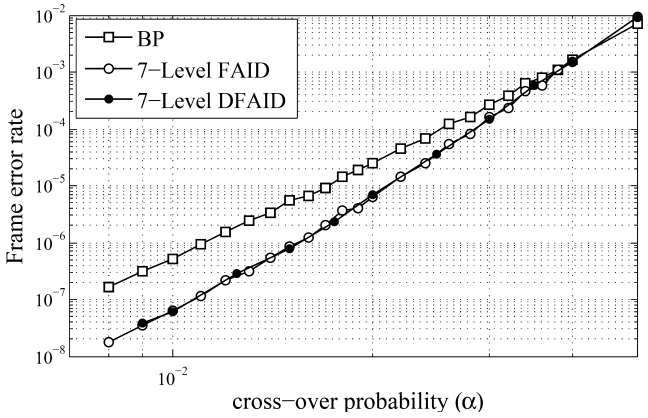


Figure A.3: Frame error rate performance comparison of Belief Propagation (BP), Finite Alphabet Iterative Decoder (FAID) and Decimation-enhanced FAID (DFAID) on the (155,93) Tanner code

Note that the above conditions on G are satisfied by most practical codes. While the theorem pertains to only a specific error pattern, a similar statement can be proven in the same spirit for any given error pattern by assuming certain conditions on its neighborhood.

Numerical results on the well-known (155,93) Tanner code are provided in Fig. A.3 in order to demonstrate that the above designed decimation rule maintains the good performance of the orignial FAID. The decimation-enhanced FAID was run using $N_d = 4$ and all decoders used a maximum of 100 iterations. On the Tanner code, with $N_d = 1$, the decimation-enhanced FAID corrects all 5 errors within 10 iterations (after decimation) whereas the 7-level FAID requires 15 iterations. At the same time, we see that decimation-enhanced FAID performs just as good as the 7-level FAID (which was known beforehand to surpass BP).

A.4 Provable guaranteed error correction using DFAIDs

Using the 7-level DFAID comprising of the decimation rule β defined by Ξ in Table A.2 and the 7-level FAID defined by Table A.1, we now present a methodology to analyze a given error pattern and based on the analysis, derive explicit conditions on the Tanner graph that ensures a guaranteed correction of the error pattern. Using this method, we present the analysis for the case of $t = 4$ and $I = 3$ where I is the number of iterations run by MP decoder on the residual graph.

The analysis for guaranteed error correction under decoding can be categorized into three main steps: 1) Analyzing the decimation of error nodes and deriving conditions on the neighborhood of the error pattern such that no error node gets decimated, 2) Analyzing the decimation of correct nodes in the graph, and examining what the subsequent residual graph at the end of decimation procedure for the given error pattern, and finally 3) examining whether the error nodes get corrected. We describe each of these steps in detail.

A.4.1 Analyzing decimation of error nodes

We previously established that as a necessary condition, no error node must be decimated for successful decoding. Therefore, the first step in proving the guaranteed correction of all error patterns of weight t on a graph G, is to show that for every t-error pattern, none of the error nodes are decimated provided that graph G satisfies certain topological conditions. These topological conditions are either known or derived during the analysis of a given error pattern. The knowledge of these topological conditions can be considered as compromise between knowing the exact neighborhood of the error pattern, and assuming that there is no knowledge on the neighborhood. This was illustrated in the previous section using the motivating example of the 4-error pattern on the 8-cycle, where conditions on the neighborhood were provided so that no error node gets decimated. We shall now describe this step in a more general setting that is applicable to any given error patten.

Given a t-error pattern that is decoded on graph G, in order to examine whether or not an error node v_i gets decimated, we need to determine the triples of messages that node v_i can receive at the end of the third iteration, and then check if that triple is included in the set Ξ. This can be done by analyzing the message-passing on the computation tree $\mathcal{T}_i^3(G)$ enumerated for three iterations. In principle, we would have to check all possible triples of messages that node v_i can receive at the end of the third iteration. However, due to the monotonicity property (Property 3) of the decimation rule β, it is sufficient to just determine the triple of messages with the least possible values that node v_i can receive at the end of third iteration, which we also refer to as *worst-case messages*. If it is determined that the triple of worst-case messages that node v_i can receive at the end of the third iteration is excluded from the set Ξ, then v_i is not decimated.

The requirement of only determining the triple of worst-case messages is a crucial aspect in this part of the analysis as it allows us to operate on a much simpler computation tree than the exact computation tree $\mathcal{T}_i^3(G)$ for the analysis if we can ensure the the message-passing on the simplified tree provides the same worst-case messages that the exact computation tree would. This simplified tree which we shall refer to as *worst-case minimal computation tree* will be defined shortly. But before we delve further, we provide the definition of a subtree and additional notations.

Definition 9 *A computation subtree is a subset of the nodes in the computation tree $\mathcal{T}_i^k(G)$ that forms a tree with a variable node in $\mathcal{T}_i^k(G)$ as its root.*

Let V^1 denote the set of error nodes (variable nodes) in H. We now define the worst-case minimal computation tree corresponding to a given error pattern $\mathbf{e}$ as follows. Recall that the zeroth level of the tree $\mathcal{T}_i^k(G)$ contains the root while the k^{th} level contains all the leaf nodes of the tree.

Definition 10 *The worst-case minimal computation tree of an error node $v_i \in V^1$ corresponding to an error pattern $\mathbf{e}$, denoted by $\mathcal{S}_i^3$, is a computation tree, that has the minimal number of nodes for which the worst-case messages are received at the root.*

Recall that the zeroth level of the tree $\mathcal{T}_i^k(G)$ contains the root, the first level contains the variable nodes that send outgoing messages towards the root, and the k^{th} level contains

all the leaf nodes of the tree. The worst-case minimal computation tree $\mathcal{S}_i^3$ can be obtained using the following steps.

1. For the given error pattern and with error node v_i being the root, draw the branches arising only from the check nodes in H and the error nodes at each level starting from the zeroth level. Leave the branches emanating from the correct nodes at each level. For instance, if the error node v_i is an isolated error node which does not share a check with any other error node, then in the first level we have only correct nodes, and we cannot proceed further.

2. Go to the first level, and examine the correct nodes on each branch in this level. We know that in the first level, by virtue of $\Phi_v(+\mathrm{C}, -L_2, -L_2) = -L_1$, the worst case message it can send is $-L_1$. Then examine what v_i receives and if the triple does not belong to Ξ, then we have obtained the worst-case minmal computation tree and proved that the error node is not decimated. Otherwise, consider all possibilities of whether this correct node can be connected either two checks in H, or just one check in H and the other being outside H, or both check being outside H. For each possibility, we need to verify that such a connection is topologically possible. For instance, if the girth is 8, we need to ensure such a connection does not introduce a six-cycle. In this manner, we can also add constraints on the neighborhood to forbid the correct node to be connected to particular checks in H. Each valid possibility leads to a subtree of depth 2, which shall be referred to as *first-level candidate subtree*.

3. For each first-level candidate subtree, draw the branches arising from check nodes in H and the error nodes. Repeat the above step for the correct nodes in the second level and each possibility is referred to as *second-level candidate subtree*. Once this is done, this automatically determines the variable nodes in the third level. Compute the messages received at the root.

4. If there are multiple candidate subtrees at each level, choose the first-level subtrees and second-level subtrees such that the root receives the worst possible messages.

Let us revert back to the example of the 4-error pattern on the 8-cycle and illustrate this approach in a general manner using Fig. A.4. Let H denoted the subgraph induced by the error pattern. Recall that the error nodes are denoted by black circles, and the correct nodes are depicted by white circles shown in the figure. Let us determine the minimal worst-case computation tree for node v_1.

Starting with the root v_1, we draw its branches. In the first level, let us assume that we don't know the connections of v_5 but we can proceed with v_2 and v_4. In the second level, we don't know the connections of v_6 and v_8. Now we consider different candidate subtrees for the correct nodes. But if the graph G must have girth-8, then nodes v_6 can only be connected to one other degree-one check of H. Same for v_8. Therefore, the subtrees chosen are the worst-case second-level subtrees. Let us also assume that v_5 sends the worst possible message regardless of its connections which is $-L_1$. The resulting tree depicted in Fig. A.4 is the minimal worst-cast computation tree $\mathcal{S}_1^3$ of node v_1.

We then see that the root v_1 receives $-L_1$ on all three of its edges. Since $\beta(-\mathrm{C}, -L_1, -L_1, -L_1) = 0$, this means that if H is contained in a girth-8 Tanner graph, then v_1 will not get decimated.

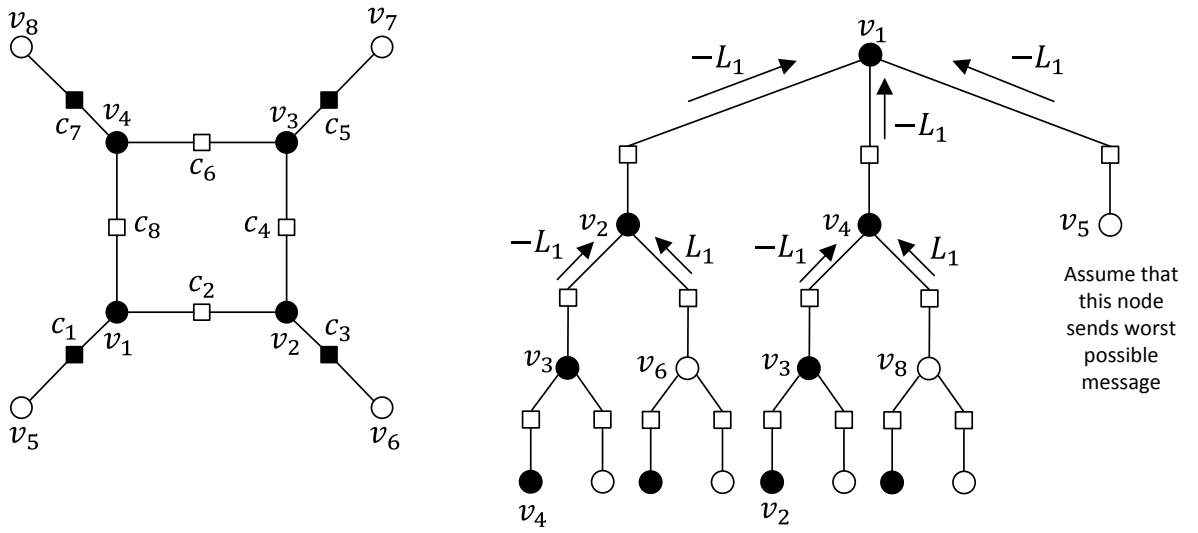


Figure A.4: Subgraph induced by the 4-error pattern and its corresponding minimal worst-case computation tree $\mathcal{S}_1^3$

By using the symmetry of H, we have just proven that no error node in H gets decimated. However, note that in general such symmetry may not exist for other error patterns. For such a case, we must also determine the minimal worst-case computation trees of multiple error nodes in the error pattern in order to ensure that no error node is decimated.

While we illustrated the approach for a particular 4-error pattern, we must repeat the same for all remaining 4-error patterns in order to ensure no error node in any 4-error pattern gets decimated. This is a necessary condition as we are subsequently required to prove that every 4-error pattern gets corrected. Fig. A.5 shows the subgraphs of all possible 4-error patterns.

By repeating the above procedure for all the remaining 4-error patterns, we have proved the following lemma.

Lemma 2 *If the graph Tanner graph G of a column-weight-three code has girth-8 and does not contain a $\mathcal{T}(6,2)$ trapping set, then for any 4-error pattern, no error node will get decimated by the 7-level DFAID.*

It is evident from the above procedure, that this analysis can be done for any given error pattern of weight t. Once we have proven that for every error pattern of weight t, none of the error nodes gets decimated, we now proceed to the next step which is to prove that they can be corrected. The link to proving their correction is to analyze the neighboring correct nodes and examine under what conditions of the neighborhood will they get decimated.

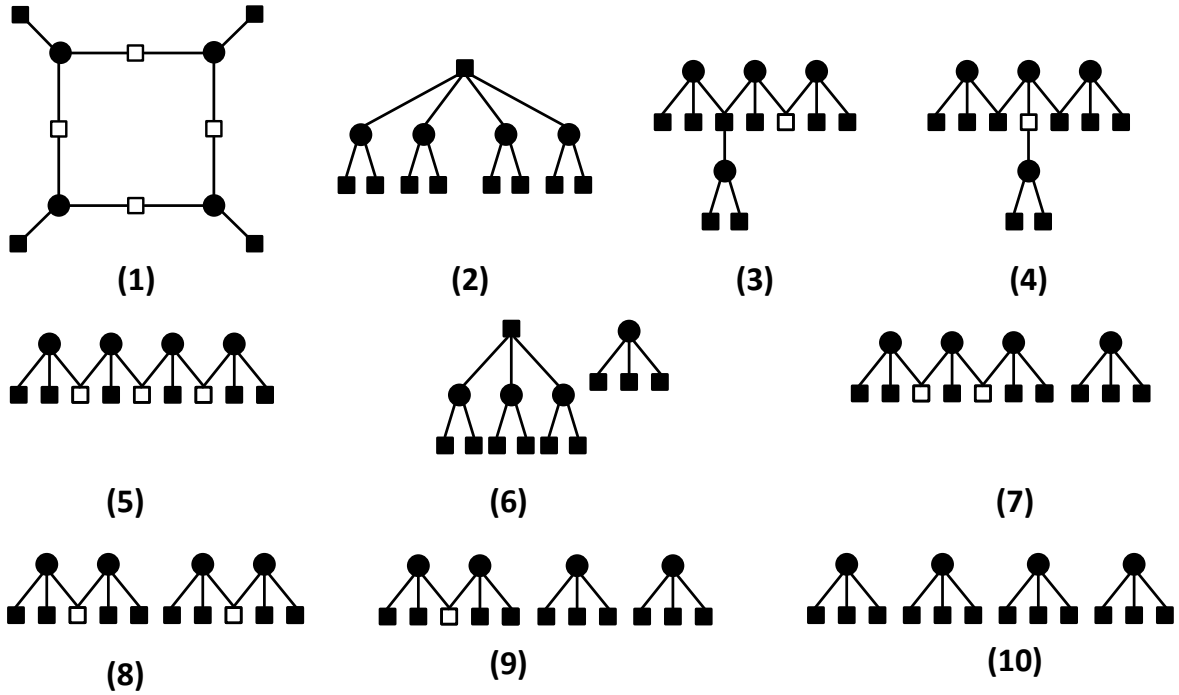


Figure A.5: Subgraphs of all possible 4-error patterns with girth-8

A.4.2 Analyzing decimation of correct nodes and proof of error-correction

Now, we want to examine whether the given error-pattern gets corrected. In order to do this, we once again draw the computation tree with an error node as its root, for k iterations, and determine what messages it will receive. If it receives messages that don't correct the node, then this means that either constraints have to be placed on its neighborhood, or certain correct nodes have to be decimated. We shall explain once again using the example of 4-error pattern on the 8-cycle shown in Fig. A.4. For sake of exposition, we shall present the description using our test case of $t = 4$ and $I = 3$ iterations.

Consider the computation tree of node v_1 for three iterations. Assuming none of the correct nodes are decimated in the tree, compute the different possible triples of messages received by node v_1 at the end of the third iteration. Observe that even if node v_5 send an L_3, v_1 would receive $-L_1$ on its remaining edges. Assuming that we use the same decision rule that is used in FAIDs, which is the sign of the total sum of the indices plus channel value $\pm$C, then the triple $(L_3, -L_1, -L_1)$ with channel value $-$C will still not correct node v_1. This mean nodes v_6 and v_8 must be decimated.

We now examine under what conditions will the nodes v_6 and v_8 get decimated. Recall that V^1 denotes the set of error nodes (variable nodes) in H. Also let V^2 denote the set of correct nodes outside the induced subgraph of the error pattern that share a check with an error node in V^1. In other words, V^2 is the set of all correct nodes at a distance of two from an error node. This can be done using the following steps.

1. First we analyze whether node v_6 gets decimated in the first round of decimation. This can be done by once again drawing the computation for three iterations using the correct node at its root, and use the same procedure that was outlined for the error nodes, except that now we determine the conditions of the neighborhood so that root receives strong enough good messages to get decimated by the β rule.

2. If node v_6 gets decimated, we are done. Else, this means that node v_6 will not get decimated in the first round. But it can still get decimated in the subsequent rounds of decimation depending on whether its nearest neighboring correct nodes get decimated in the first round or not. If even those nearest neighboring correct nodes do not get decimated in the first round, we then continue to expand by examining nearest neighboring correct nodes of those correct nodes in V^2. This continues until either we eventually reach correct nodes that get decimated in the first round, or we obtain the final residual graph. We then check the messages passed along the new computation tree of root node v_i where certain nodes may be decimated to see if it gets corrected or not.

3. During this analysis, we can forbid certain topological connections emanating from certain correct nodes to limit the amount of expansion we need to do, which in turn eliminates certain subgraphs.

Let $\mathcal{G}(N, M)$ denote a forbidden graph with N variable nodes and M check nodes. Fig. A.6 shows the forbidden subgraphs that were obtained during the analysis of the 4-error

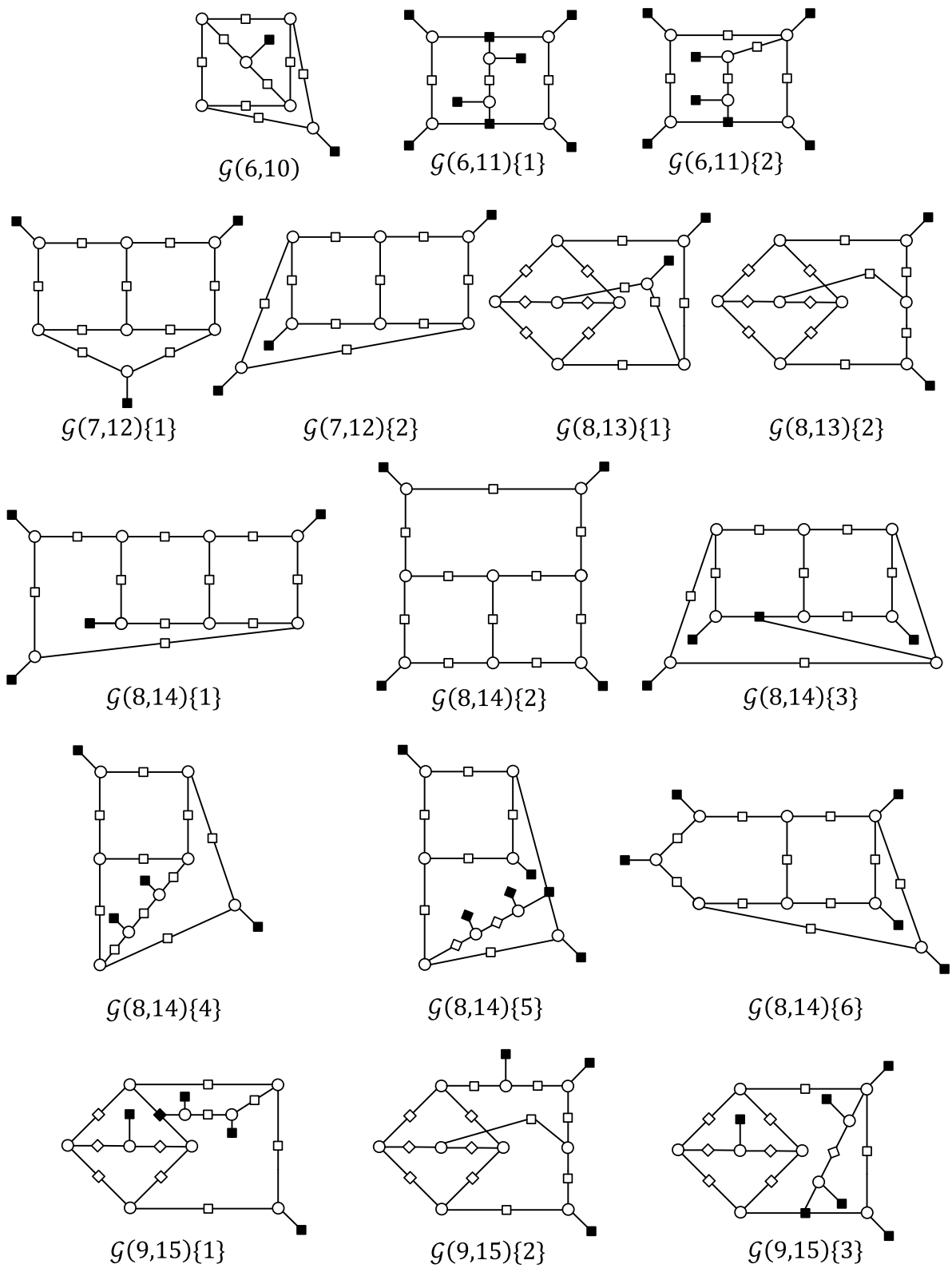


Figure A.6: Forbidden subgraphs belonging to set $\mathscr{F}$ for guaranteeing correction of the $t = 4$ errors in $I = 3$ iterations

pattern on the 8-cycle. Note that {} is used to distinguish between two graphs with the same number of variable nodes and check nodes. Let $\mathscr{F}$ denote the set of all forbidden graphs depicted in Fig. A.6. For the 4-error pattern, we can prove the following theorem.

Theorem 3 *If the graph G of a column-weight-three code has girth-8, and does not contain any subgraphs belonging to $\mathscr{F}$, then the DFAID requires at most two decimations, and three iterations after decimation to correct the 4-error pattern whose induced subgraph forms an 8-cycle.*

The complete proof for the case of $t = 4$ is currently in progress, but we believe the list of forbidden subgraphs to avoid will not change. The complete proof for the theorem will be provided in a revised draft of this paper.

A.5 Conclusions and Remarks

We have proposed decimation-enhanced FAIDs, and illustrated how decimation can be used as tool to make the decoder more amenable to analysis. We have presented a methodology to analyze the message-passing of DFAIDs for a finite number of iterations, and prove the guaranteed error-correction capability using the analysis on decimation, and deriving conditions on the neighborhood of the subgraph induced by the error pattern. The completion of the proof of the theorem that DFAIDs can guarantee correction of $t = 4$ errors and $I = 3$ iterations is currently under progress. Then the next step would be to design codes which do not contain the forbidden graphs to highest possible code rate, and then compare their error-rate performance with other decoders under a few number of iterations. The broader goal is to be able to utilize such a code and the DFAID decoder in the TK fault-tolerant memory model in order to be able to realize practical finite-length fault-tolerant memories.

This work was supported by the Seventh Framework Program of the European Union, under Grant Agreement number 309129 (i-RISC project).

Bibliography

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

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

[3] N. Wiberg, *Codes and Decoding on General Graphs*. PhD thesis, Linkoping University, Sweden, 1996.

[4] C. A. Kelly and D. Sridhara, "Pseudocodewords of Tanner graphs," *IEEE Trans. Inf. Theory* , vol. 53, pp. 599–618, Feb. 2001.

[5] P. Vontobel and R. Koetter, "Graph-Cover Decoding and Finite-Length Analysis of Message-Passing Iterative Decoding of LDPC Codes," `http://arxiv.org/abs/cs/0512078`.

[6] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke, Finite-length analysis of low-density parity-check codes on the binary erasure channel, *IEEE Trans. on Inform. Theory*, vol. IT48, no. 6, pp. 1570-1579, 2002.

[7] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conf. on Commun., Control and Computing*, 2003.

[8] S. K. Chilappagari, S. Sankaranarayanan, and B. Vasic, "Error floors of LDPC codes on the binary symmetric channel," *Proc. IEEE Int. Conf. on Commun.*, Istanbul, Turkey, pp. 1089–1094, Jun. 2006.

[9] V. Chernyak, M. Chertkov, M. Stepanov, and B. Vasic, Instanton method of posterror-correction analytical evaluation, in *Proc. IEEE Inform. Theory Workshop*, pp. 220-224, Oct. 2004.

[10] D. Burshtein and G. Miller, "Expander arguments for message-passing algorithms," *IEEE Trans. Inf. Theory*, vol. 47, Issue 2, pp. 782–790, Feb. 2001.

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

[12] S. K. Chilappagari, D. V. Nguyen, B. Vasic, and M. W. Marcellin, "Error correction capability of column-weight-three LDPC codes under the Gallager A algorithm-Part II," *IEEE Trans. Inf. Theory.*, vol. 56, no. 6, pp. 2626-2639, Jun. 2010

[13] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic,"FAID, Part 1: Decoding beyond BP on the BSC," *IEEE Trans. Commun.*, vol.61, no.10, Oct. 2013.

[14] D. Declercq, L. Danjean, E. Li, S. Planjery, and B. Vasic,"Finite alphabet iterative decoding (FAID) of the (155,64,20) Tanner code," in *Proc.* 6^{th} *Int. Symp. on Turbo codes (ISTC'10)*, Sept. 2010.

[15] A. Montanari, F. Ricci-Tersenghi, and G. Semerjian, "Solving constraint satisfaction problems through belief propagation-guided decimation," in *Proc. Allerton Conf. on Commun.*, 2007.

[16] A. G. Dimakis, A. A. Gohari, M. J. Wainwright, "Guessing Facets: Polytope Structure and Improved LP Decoder," *IEEE Trans. Inf. Theory*, vol. 55, Issue 8, pp. 3479–3487, Aug. 2009.

[17] M. Chertkov, "Reducing the error floor," in *Proc. IEEE Inf. Theory Workshop*, pp. 230–235, Sept. 2007.

[18] B. J. Frey, R. Koetter, and A. Vardy, "Signal-space characterization of iterative decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 766–781, Feb. 2001.

[19] "Error Floors of LDPC Codes." [Online]. Available: `http://www.ece.arizona.edu/~vasiclab/Projects/CodingTheory/ErrorFloorHome.html`

[20] M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," *Proc. 5th Int. Symp. Commun. Theory App.*, Ambleside, England, Jul. 2001.