Guaranteed Error Correction Capability of Codes on Graphs

Shashi Kiran Chilappagari, Bane Vasic and Michael W. Marcellin
Department of Electrical and Computer Engineering,
University of Arizona,
Tucson, AZ 85721, U.S.A.
email:{shashic,vasic,marcellin}@ece.arizona.edu

***Abstract*— The guaranteed error correction capability of left regular LDPC codes under different hard decision decision algorithms is investigated. A summary of recent results relating the column weight and girth of the Tanner graph to the guaranteed error correction capability is provided. The intuition behind expander based arguments and their potential to derive new results for column-weight-three codes is provided.**

I. Introduction

Low-density parity-check (LDPC) codes [1] have attracted significant attention due to their capacity approaching performance under low-complexity sub-optimal decoding algorithms. Many properties of ensembles of regular and irregular LDPC codes under such low-complexity decoding algorithms in the asymptotic limit of the code length $n \to \infty$ are now well understood (see [2], [3], [4]). Broadly speaking, the properties of LDPC codes have been explored via two avenues: (1) density evolution to compute thresholds and analyze the performance of LDPC codes under various message passing algorithms [2] and (2) expander graph based arguments to derive bounds on guaranteed error correction capability [5]. In this paper, we review the expander graph based approach with focus on hard decision decoding algorithms for transmission over the binary symmetric channel (BSC). Specifically, we consider binary message passing algorithms, namely Gallager A/B algorithms, and bit flipping algorithms (serial and parallel). This paper aims at providing an intuition underlying the expander based arguments rather than proofs. The interested reader is referred to [6], [7], [8], [9] for more rigorous presentation of the results.

Gallager [1] showed that the minimum distance of ensembles of (d_v, d_c) regular LDPC codes with $d_v \geq 3$ grows linearly with the code length. This implies that under maximum likelihood (ML) decoding, these codes are capable of correcting a number of errors linear in the code length. Gallager proposed two low complexity sub-optimal binary message passing algorithms, namely Gallager A and Gallager B algorithms, for decoding over the BSC. He showed that the bit error probability under these algorithms approaches zero, whenever we operate below the threshold, but the correction of a number of errors linear in the code length was not shown. Zyablov and Pinsker [10] analyzed regular LDPC code ensembles with $d_v \geq 5$ and showed that almost all the codes in these ensembles can correct a linear number of errors in the code length under the bit flipping decoding algorithms. Sipser and Spielman [5] proved similar results using expansion arguments. Recently, Burshtein [11] showed that the ensemble of regular codes with $d_v = 4$ are also capable of correcting a linear number of errors under the parallel bit flipping algorithm. Burshtein and Miller [12] used expander based arguments to show that message passing algorithms for irregular code ensembles are also capable of correcting a linear number of errors in the code length when the degree of each variable node is at least six.

II. The Approach: Why and How

A. Bit Flipping, Expansion and Error Correction

Expander graph based arguments provide a convenient way to analyze the guaranteed error correction capability of LDPC codes with $d_v \geq 5$ under the bit flipping algorithms. These arguments show that the bit flipping decoders can correct a linear number of errors in the code length when the underlying Tanner graph [13] is a good expander. The Tanner graphs of codes in the ensembles with $d_v \geq 5$ meet this requirement with very high probability. As Richardson and Urbanke note in [14], an advantage of expander graph based arguments is their applicability to finite length codes. However, a limitation this approach is that the bounds derived on the fraction of nodes having the required expansion, using random graph arguments, is very small, and therefore requiring the codes to have large length to guarantee correction of a even a small number number of errors. As an illustrative example, consider the $(5, 6)$ regular LDPC code ensemble. In [14], it was shown that a randomly chosen code in this ensemble has minimum distance of at least 3% of the code length with high probability, while the fraction of nodes having required expansion with high probability is about 3.375×10^{-11}. It is also worth noting that determining the expansion of a given graph is known to be non-deterministic polynomial-time (NP) hard [15] and the spectral gap methods can only guarantee an expansion factor of $1/2$ or less. While a random graph is a good expander with high probability, explicit construction of codes having the required expansion is not known for small values of d_v and small code lengths.

One observes that the expansion based arguments rely heavily on properties of random graphs and hence do not lead to explicit construction of codes. However, if the expansion

properties can be related to the parameters of the Tanner graph, such as girth and column weight, which are known or can be easily determined, then the bounds on guaranteed error correction capability can be established as function of these parameters. This is the general idea of our approach that we review in this paper.

B. The Case of Column-Weight-Three Codes

LDPC codes with $d_v = 3$ are of special importance as their decoders have low-complexity and are interesting for a variety of practical applications. From the above overview of literature, it is clear that the expansion based arguments are not applicable to code with $d_v = 3$. This can be attributed to the fact the Tanner graphs of the codes in this ensemble cannot achieve the expansion required by the expansion based arguments. Recently, we showed that while the minimum distance of codes in the (d_v, d_c) regular LDPC codes ensemble with $d_v = 3$ grows linearly with the code length, the error correction capability under hard decision decoding algorithms grows only logarithmically with the code length (see [9] for a proof). The presence of cycles in the Tanner graph of the code leads to decoder failures and since the girth grows only logarithmically with the code length, correction of linear fraction of errors cannot be achieved. It is worth noting here that for the more complex linear programming (LP) decoding also, it has not been shown that codes with $d_v = 3$ can correct a fraction of errors. Design of decoding strategies that can ensure the correction of linear number of errors for these codes is arguably one of the most important open problems in the theory of iterative decoding.

C. Better Codes and Better Decoders

Since the sub-optimal hard decision decoding algorithms for LDPC codes are far worse compared to the maximum-likelihood decoding, we introduce a framework in which the gap can be bridged by considering improved code construction techniques and more complex decoders. While soft decision decoding algorithms exhibit superior performance, their analysis is complicated by implementation nuances such as numerical precision of the messages. In order to alienate such effects and study only the effect of the code's structure on its performance, we restrict our attention to the BSC and decoding algorithms with finite size alphabet for messages. Given a code and a decoding algorithm, one can gain an improvement in the performance either by constructing a better code or by designing a better decoding algorithm or a combination of both. An understanding of the failures of the decoding algorithm as a function of the code's structure is central to both the approaches. The knowledge of underlying topological structures in the Tanner graph that lead to decoder failures can be used to construct better codes that avoid such harmful configurations. This knowledge can also be exploited to design decoder algorithms that can overcome such decoder failures.

While the failures of iterative decoding over the binary erasure channel (BEC) are completely characterized by combinatorial structures known as stopping sets [16], the same level of maturity in understanding the decoding failures of other decoders has not been attained yet. Two approaches that have met with success in the finite length analysis of a given code under iterative decoding are the methods based on trapping sets [17] and pseudo-codewords [18]. Roughly speaking, a trapping set is a set of variable nodes that are decoded wrongly over the iterations. The concept of pseudo-codewords arises from the notions of computation trees [19] and graph cover decoding [18] (see [20] for an exhaustive list of references in this area).

For the case of LDPC codes with $d_v = 3$, we classify the failures of the Gallager A decoder when the Tanner graph satisfies certain conditions. We then show how message passing algorithms with larger alphabet can be overcome these failures.

III. Summary of Results

The results presented in this section were presented first in [9], [6], [7], [8]. The reader is referred to Appendix for notation and overview of background material.

A. Girth and Expansion

It is known that if the Tanner graph G of an LDPC code of length n is an $(d_v, d_c, \alpha, (3/4 + \epsilon)d_v)$ expander, then the parallel bit flipping algorithm can correct all error patterns with up to $\alpha(1 + 4\epsilon)n/2$ errors. Hence, if we know the size of variable node sets that are guaranteed to expand by at least $(3/4)d_v$, we can obtain a lower bound on the guaranteed error correction capability under the parallel bit flipping algorithm. We find these bounds as a function of d_v and the girth of the Tanner graph of the code. Note that such bounds are already known for minimum distance and minimum stopping sets size (see [13] and [21] respectively). In the language of expansion, bounds on the size of variable node sets that expand by at least a factor of $d_v/2$ are known. Such bounds can be derived for an expansion factor of $(3/4)d_v$.

Error correction capability grows exponentially with girth: Let $G = (V \cup C, E')$ be a $d_v \geq 4$-left-regular Tanner graph G with $g(G) = 2g'$. Then for all $k < n_0(d_v/2, g')$, any set of k variable nodes in G expands by a factor of at least $(3/4)d_v$, where

$$n_0(\overline{d}, g) = n_0(\overline{d}, 2r+1) \quad = \quad 1 + \overline{d}\sum_{i=0}^{r-1}(\overline{d}-1)^i, \ g \text{ odd}$$

$$n_0(\overline{d}, g) = n_0(\overline{d}, 2r) \quad = \quad 2\sum_{i=0}^{r-1}(\overline{d}-1)^i, \ g \text{ even.}$$

To put the above result in perspective, we observe that the minimum distance (as well as minimum stopping set size) of a d_v regular code with girth $2g'$ is at least $n_0(d_v, g')$. Hence, for ML decoding on the BSC, we see that the error correction capability is $\lfloor n_0(d_v, g') - 1 \rfloor /2$.

Fixed and trapping sets for bit flipping decoders: In parallel bit flipping decoder, one observes that the decoder runs till no variable is involved in more unsatisfied checks than satisfied checks. This however, does not guarantee that the decoder finds a codeword. It can get "trapped" in a set of variable nodes i.e., either the decoder does not find any variable nodes to flip or it exhibits an oscillatory behavior.

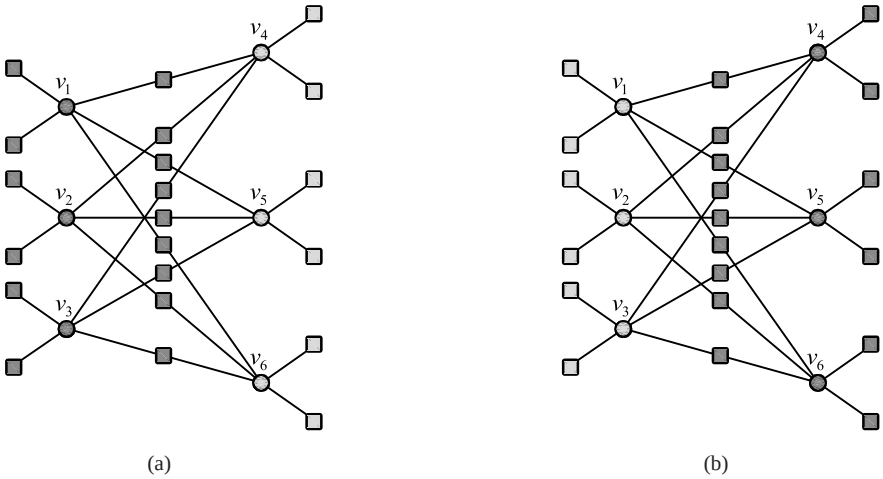

Fig. 1. Illustration of oscillations in the bit flipping decoder. In this figure, ● represents a correct variable node, ● represents a corrupt variable node, ■ represents an unsatisfied check node and ■ represents a satisfied check node (a) The set of corrupt variables $\{v_1, v_2, v_3\}$ at the beginning of the first iteration. The check node neighbors of v_1, v_2 and v_3 are all unsatisfied (b) The set of corrupt variables $\{v_4, v_5, v_6\}$ at the beginning of the second iteration

The set of variable nodes that are not eventually corrected by the decoder is called a trapping set. If the set of corrupt variables remains the same after every iteration, we call that set of corrupt variables as a fixed set.

Fig. 1 illustrates oscillations in a code with $d_v = 5$ and girth 8. Note here that $\{v_1, \ldots, v_6\}$ is a trapping set and a fixed set. This code cannot correct three errors. For correction of three errors, an expansion factor of $(3/4)d_v$ for all variable sets of size less than or equal to six would have implied that any set of six variable nodes should have at least 23 neighboring checks. However, the set of six variable nodes in Fig. 1 has 21 neighboring check nodes and fails to expand by the required amount. This example illustrates that the expansion factor of $(3/4)d_v$ cannot be improved substantially and in order to guarantee better performance for the same expansion, we should consider better decoding algorithms.

Potential fixed sets from cage graphs: A (d, g)*-cage graph*, $G(d, g)$, is a d-regular graph with girth g having the minimum possible number of nodes. Fig. 2 shows how a potential fixed set can be obtained from a cage graph. The cage graph depicted in Fig. 2(a) is the $(3, 5)$ cage graph (it is the unique $(3, 5)$ cage graph), also known as the Peterson graph. Fig. 2(b) shows a potential fixed set for a code with Tanner graph of column weight five and girth ten.

B. Column-Weight-Three LDPC Codes

Cycles cause failures: The best achievable expansion by an LDPC code with Tanner graph G is at most $1 - 1/d_v$. For $d_v = 3$ it can be seen that this maximum value is $2/3$ and hence column-weight-three LDPC codes cannot achieve the expansion required to guarantee correction of a linear number of errors. However, closer inspection of the properties of these codes allows us to make stronger statements. In fact, it can be shown that given any $\alpha > 0$, at sufficiently large values of n, no code with $d_v = 3$ can correct all error patterns with up to αn errors i.e., no code in the column-weight-three LDPC code ensemble can correct a linear number of errors in the code length under bit flipping as well as Gallager A algorithms.

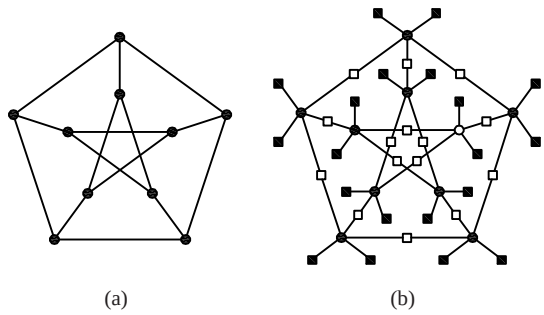

Fig. 2. (a)The $(3, 5)$ cage graph, also known as the Peterson graph (b) A potential fixed set of size 10 for a code with Tanner graph of column weight 5 and girth 10

This result stems from the following observations. Consider any set of variable nodes involved in a cycle and assume that only these variables are in error. In the induced subgraph of this set of variable nodes, each variable is connected to two degree two checks and one degree one check. One can see that each variable in this cycle is involved in more satisfied constraints than unsatisfied constraints. Hence, if no variable outside this set shares two degree-one checks in this subgraph, then the bit flipping decoder does not find any variable node to flip and the decoder gets trapped in these variable nodes. So, errors in cycles lead to decoder failures for column-weight-three codes. Since the length of smallest cycle in the Tanner graph grows only logarithmically with the code length, we can see that the code can correct at most logarithmic number of errors in the code length.

For the Gallager A algorithm it can be shown that similar results hold good. Moreover, it can be proved that the Gallager A algorithm for column-weight-three code with Tanner graph of girth $g \geq 10$ can correct all error patterns with up to $g/2 - 1$ errors in $g/2$ iterations. The preceding statement provides sufficient conditions to correct all error patterns with up to k errors for $k \geq 4$. To correct three errors, we additionally need to avoid some trapping sets (see for details).

To summarize: cycles are the main causes of decoder

failures for column-weight-three codes and since the girth cannot be increased beyond a certain point for a code of given length and rate, the performance of column-weight-three codes is inherently restricted by the best achievable girth.

IV. Work in Progress

Codes avoiding trapping sets exhibit superior performance: While high girth codes show superior performance in general, the results presented here indicate that girth alone does not guarantee good error correction capability. Since a given rate and code length place a fundamental limit on the best achievable girth, one needs to construct Tanner graph avoiding small trapping sets in order to build good codes. A natural way to proceed would be to analyze all error patterns with up to certain errors and identify the troublesome configurations. A code can then be constructed by building a Tanner graph free of all such harmful structures. However, while this problem can be easily stated, the answer is not known for a majority of the cases.

Two bit decoders can correct more errors: For column-weight-three codes we have seen that cycles pose a problem for the single bit decoders. It is easy to come up with a stronger decoding algorithm that can overcome a cycle. But there can exist other error configurations not involved in cycles that can lead to decoder failures. It is in general very difficult to characterize all error patterns up to certain weight. However, we claim that that for Tanner graphs with expansion close to the limit of $2/3$, the only variable nodes that the Gallager A decoder fails to correct are those involved in cycles (a formal proof for this claim is currently being investigated). The Gallager A fails to correct errors in a cycle as it always receives two incorrect messages and only one correct message. For Tanner graphs that have expansion close to $2/3$, it can be proved that all other variable nodes receive at most one incorrect message. If a decoding rule that can differentiate the correct message and incorrect message can be formulated, it is easy to see that errors in cycle can also be corrected leading to superior performance. Even decoders with erasure in the message alphabet (see [2]) can be used in such scenarios. Our preliminary investigation into this problem shows that Gallager A followed by simple two-bit decoders can correct a large number of errors. It remains to see if it can be proved that there exists a multi-bit decoder that can correct a linear number of errors for column-weight-three codes. The more powerful two-bit decoders can also be used to analyze higher column-weight-codes to get better bounds on guaranteed error correction capability.

Appendix

We provide notation and background material in this appendix.

A. LDPC Codes

We adopt the standard notation in graph theory (see [22] for example). $G = (U, E)$ denotes a graph with set of nodes U and set of edges E. When there is no ambiguity, we simply denote the graph by G. An edge e is an unordered pair (u_1, u_2) of nodes and is said to be incident on u_1 and u_2. Two nodes u_1 and u_2 are said to be adjacent (neighbors) if there is an edge $e = (u_1, u_2)$ incident on them. The order of the graph is $|U|$ and the size of the graph is $|E|$. $N(u)$ denotes the set of neighbors of u. The degree of u, $d_u = |N(u)|$, is the number of its neighbors. A node with degree one is called a leaf node. A graph is d-regular if all the nodes have degree d. The average degree $\overline{d}$ of a graph G is the average over $d(u)$ for all $u \in U$. The girth $g(G)$ of a graph G, is the length of smallest cycle in G. $H = (V \cup C, E')$ denotes a bipartite graph with two sets of nodes; variable (left) nodes V and check (right) nodes C and edge set E'. Nodes in V have neighbors only in C and vice versa. A bipartite graph is said to be d_v-left-regular if all variable nodes have degree d_v, d_c-right-regular if all check nodes have degree d_c and (d_v, d_c) regular if all variable nodes have degree d_v and all check nodes have degree d_c. The girth of a bipartite graph is even.

LDPC codes [1] are a class of linear block codes which can be defined by sparse bipartite graphs [13]. Let G be a bipartite graph with two sets of nodes: n variable nodes and m check nodes. This graph defines a linear block code $\mathcal{C}$ of length n and dimension at least $n - m$ in the following way: The n variable nodes are associated with the n coordinates of codewords. A vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ is a codeword if and only if for each check node, the modulo two sum of its neighbors is zero. Such a graphical representation of an LDPC code is called the Tanner graph of the code [13]. An (n, d_v, d_c) regular LDPC code has a Tanner graph with n variable nodes each of degree d_v (column weight) and nd_v/d_c check nodes each of degree d_c (row weight). This code has length n and rate $r \geq 1 - d_v/d_c$ [1]. The adjacency matrix of G gives a parity check matrix of $\mathcal{C}$.

B. Decoding Algorithms

We now describe a simple hard decision decoding algorithm known as the parallel bit flipping algorithm [10], [5] to decode LDPC codes. As noted earlier, each check node imposes a constraint on the neighboring variable nodes. A constraint (check node) is said to be satisfied by a setting of variable nodes if the sum of the variable nodes in the constraint is even; otherwise the constraint is unsatisfied.

Parallel Bit Flipping Algorithm

- In parallel, flip each variable that is in more unsatisfied than satisfied constraints.
- Repeat until no such variable remains.

Note that if d_v is even then a variable node that receives $d_v/2$ flip messages is not flipped. A serial version of the algorithm is also defined in [5] and all the results in this paper hold for the serial bit flipping algorithm also. The bit flipping algorithms are iterative in nature but do not belong to the class of message passing algorithms (see [12] for an explanation). In general, we assume that the parallel bit flipping algorithm is run for a maximum number of M iterations.

Gallager A algorithm: The Gallager A algorithm is a binary message passing algorithm [1]. Every round of message passing (iteration) starts with the variable nodes sending messages to their neighboring check nodes (first half of the

iteration) and ends by the check nodes sending messages to their neighboring variable nodes (second half of the iteration). Let $\mathbf{r}$, a binary n-tuple, be the input to the decoder. Let $\omega_j(v,c)$ denote the message passed by a variable node v to its neighboring check node c in j^{th} iteration and $\varpi_j(c,v)$ denote the message passed by a check node c to its neighboring variable node v. Let $\mathcal{N}(c)$ denote the set of neighbors of check node c. Additionally, let $\omega_j(v,:)$ denote the set of all messages from v, $\omega_j(v,:\backslash c)$ denote the set of all messages from v except to c, $\omega_j(:,c)$ denote the set of all messages to c. The terms $\omega_j(:\backslash v,c), \varpi_j(c,:), \varpi_j(c,:\backslash v), \varpi_j(:,v)$ and $\varpi_j(:\backslash c,v)$ are defined similarly. The Gallager A algorithm can be defined as follows.

$$\begin{aligned}
\omega_1(v,c) &= r_v \\
\omega_j(v,c) &= \begin{cases} 1, & \text{if } \varpi_{j-1}(:\backslash c,v)=\{1\} \\ 0, & \text{if } \varpi_{j-1}(:\backslash c,v)=\{0\} \\ r_v, & \text{otherwise} \end{cases} \\
\varpi_j(c,v) &= \left(\sum_{u\in\mathcal{N}(c)\backslash v} \omega_j(u,c)\right) \bmod 2
\end{aligned}$$

At the end of each iteration, an estimate of each variable node is made based on the incoming messages and possibly the received value. The codeword estimate of the decoder at the end of jth iteration is denoted as $\mathbf{r}^{(j)}$. The decoder is run until a valid codeword is found or a maximum number of iterations M is reached, whichever is earlier. The output of the decoder is either a codeword or $\mathbf{r}^{(M)}$.

C. Expansion and Error Correction Capability

Definition 1: Let $G=(U,E)$ with $|U|=n_1$. Then *every set of at most m_1 nodes expands by a factor of δ* if, for all sets $S\subset U$

$$|S|\le m_1 \Rightarrow |\{y:\exists x\in S \text{ such that } (x,y)\in E\}|>\delta|S|.$$

We consider bipartite graphs and expansion of variable nodes only.

Definition 2: A graph is a (d_v,d_c,α,δ) expander if it is a (d_v,d_c) regular bipartite graph in which every subset of at most α fraction of the variable nodes expands by a factor of at least δ.

The following fact from [5] relates the expansion and error correction capability of an (n,d_v,d_c) LDPC code with Tanner graph G when decoded using the parallel bit flipping decoding algorithm.

Fact 1: [5, Theorem 11] Let G be a $(d_v,d_c,\alpha,(3/4+\epsilon)d_v)$ expander over n variable nodes, for any $\epsilon>0$. Then, the simple parallel decoding algorithm will correct any $\alpha_0<\alpha(1+4\epsilon)/2$ fraction of errors after $\log_{1-4\epsilon}(\alpha_0 n)$ iterations..

D. Decoder Failures and Trapping Sets

For linear codes transmitted over output symmetric channels and decoded using symmetric decoding algorithms, we can assume, without loss of generality, that the all-zero-codeword is transmitted [2]. We make this assumption throughout the paper. The support of a vector $\mathbf{y}=(y_1,y_2,\ldots,y_n)$, denoted by supp$(\mathbf{y})$ is defined as the set of all variable nodes v for which $y_v\neq 0$. Since the transmitted codeword is assumed to be the all-zero-codeword, the support of the input to the decoder is simply the set of variable nodes flipped by the channel (or in other words the set of variable nodes initially in error). We adopt the the definitions of the terms eventually correct variable nodes, trapping sets and failure sets from [17].

Let $\mathbf{y}$ denote the input to the decoder and $\mathbf{y}^{(l)}$ denote the codeword estimate of the decoder at the end of lth iteration. A variable node v is said to be *eventually correct* if there exists a positive integer l_c such that for all $l\ge l_c$, $y_v^l=0$. For an input $\mathbf{y}$, the *failure set* $\mathbf{T}(\mathbf{y})$ is defined as the set of variable nodes that are not eventually correct. The decoding on the input $\mathbf{y}$ is successful if and only if $\mathbf{T}(\mathbf{y})=\emptyset$. If $\mathbf{T}(\mathbf{y})\neq\emptyset$, then we say that $\mathbf{T}(\mathbf{y})$ is a *trapping set* and supp$(\mathbf{y})$ is an *inducing set*. The size of an inducing set is its cardinality. While the above notions of inducing and trapping sets are fundamental in characterizing the decoding failures, the definitions do not lead to the formulation of necessary and sufficient conditions for a set of variable nodes to be a trapping/inducing set for the parallel bit flipping algorithm. We therefore define fixed sets, for which such conditions can be derived.

Definition 3: [9] Let $\mathcal{F}$ be the set of corrupt variable nodes at the beginning of a decoding iteration in the bit flipping (serial or parallel) algorithm. $\mathcal{F}$ is a fixed set if the set of corrupt variable nodes at the end of the iteration is $\mathcal{F}$. A vector $\mathbf{y}$ with supp$(\mathbf{y})=\mathcal{F}$ is called a fixed point.

It is clear from the above definition that if the input to the decoder is a fixed point, then the output of the decoder is the same fixed point. It follows that if $\mathbf{y}$ is a fixed point and $\mathbf{T}(\mathbf{y})\neq\emptyset$, then $\mathbf{T}(\mathbf{y})=$ supp$(\mathbf{y})$ is a trapping set as well as an inducing set. It is clear that if the Tanner graph G of a code $\mathcal{C}$ has a fixed set of size k, then the code cannot correct all error patterns with up to k errors.

Theorem 1: Let $\mathcal{C}$ be an LDPC code with d_v-left-regular Tanner graph G. Let $\mathcal{T}$ be a set consisting of V variable nodes with induced subgraph $\mathcal{I}$. Let the checks in $\mathcal{I}$ be partitioned into two disjoint subsets; $\mathcal{O}$ consisting of checks with odd degree and $\mathcal{E}$ consisting of checks with even degree. Then $\mathcal{T}$ is a fixed set for the bit flipping algorithm (serial or parallel) iff : (a) Every variable node in $\mathcal{I}$ has at least $\lceil d_v/2\rceil$ neighbors in $\mathcal{E}$, and (b) No $\lfloor d_v/2\rfloor+1$ checks of $\mathcal{O}$ share a neighbor outside $\mathcal{I}$.

To determine whether a given set of variables is a fixed set, it is necessary to not only know the induced subgraph but also the neighbors of the odd degree checks. However, in order to establish general bounds on the sizes of fixed sets given only the column weight and the girth, we consider only condition (a) of Theorem 1 which is a necessary condition. A set of variable nodes satisfying condition (a) is known as a *potential fixed set*. It is worth noting that a potential fixed set can always be extended to a fixed set by successively adding a variable node till condition (b) is satisfied.

Acknowledgment

The authors would like to thank A. R. Krishnan, D. V. Nguyen and S. Planjery for their contributions. The authors wish to acknowledge the financial support of the

NSF (Grants CCF-0634969 and ECCS-0725405) and Seagate Technology.

REFERENCES

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

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

[3] T. J. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 638–656, 2001.

[4] L. Bazzi, T. Richardson, and R. Urbanke, "Exact thresholds and optimal codes for the binary symmetric channel and Gallager's decoding algorithm A," *IEEE Trans. Inform. Theory*, vol. 50, pp. 2010–2021, 2004.

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

[6] 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," May 2008. [Online]. Available: http://arxiv.org/abs/0805.2427

[7] ——, "On the guaranteed error correction capability of LDPC codes," in *Proc. IEEE International Symposium on Information Theory (ISIT '08)*, July 2008, pp. 434–438. [Online]. Available: http://dx.doi.org/10.1109/ISIT.2008.4595023

[8] ——, "Error correction capability of column-weight-three LDPC codes: Part II," Jul 2008. [Online]. Available: http://arxiv.org/abs/0807.3582

[9] S. K. Chilappagari and B. Vasic, "Error correction capability of column-weight-three LDPC codes," *IEEE Trans. Inform. Theory*, accepted for publication. [Online]. Available: http://arxiv.org/abs/0710.3427

[10] V. V. Zyablov and M. S. Pinsker, "Estimation of the error-correction complexity for Gallager low-density codes," *Problems of Information Transmission*, vol. 11, no. 1, pp. 18–28, 1976.

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

[12] D. Burshtein and G. Miller, "Expander graph arguments for message-passing algorithms," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 782–790, 2001.

[13] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[14] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, March 2008. [Online]. Available: http://lthcwww.epfl.ch/mct/index.php

[15] N. Alon, "Spectral techniques in graph algorithms," in *Lecture Notes in Computer Science 1380*. Springer-Verlag, 1998, pp. 206–215.

[16] C. Di, D. Proietti, T. Richardson, E. Telatar, and R. Urbanke, "Finite length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, pp. 1570–1579, 2002.

[17] T. J. Richardson, "Error floors of LDPC codes," in *Proc. 41st Annual Allerton Conf. on Communications, Control and Computing*, 2003, pp. 1426–1435. [Online]. Available: http://www.hpl.hp.com/personal/Pascal_Vontobel/pseudocodewords/papers

[18] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite length analysis of message-passing iterative decoding of LDPC codes," 2005. [Online]. Available: http://arxiv.org/abs/cs.IT/0512078

[19] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Univ. Linköping, Sweden, Dept. Elec. Eng., 1996.

[20] P. O. Vontobel, "Papers on pseudo-codewords." [Online]. Available: http://www.hpl.hp.com/personal/Pascal_Vontobel/pseudocodewords/papers

[21] A. Orlitsky, R. Urbanke, K. Viswanathan, and J. Zhang, "Stopping sets and the girth of Tanner graphs," in *IEEE Int. Symp. on Infrom. Theory*, 2002, p. 2.

[22] B. Bollobas, *Extremal graph theory*. London: Academic Press Inc., 1978.