Characterization of Problematic Graphical Structures of LDPC Codes and the Corresponding Efficient Search Algorithms

by

Yoones Hashemi Toroghi, M.Sc.

A dissertation submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario
August, 2017

The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the dissertation

Characterization of Problematic Graphical Structures of LDPC Codes and the Corresponding Efficient Search Algorithms

submitted by **Yoones Hashemi Toroghi, M.Sc.**

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Professor Amir H. Banihashemi, Thesis Supervisor

Professor Bane Vasic, External Examiner

Professor Yvan Labiche, Chair,
Department of Systems and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
August, 2017

Abstract

In this thesis, we study all the problematic graphical structures which play important roles in the error floor performance and error correction capability of low-density parity-check (LDPC) codes. These graphical structures are: *leafless elementary trapping sets (LETSs)*, *elementary trapping sets (ETSs)*, *non-elementary trapping sets (NETSs)*, *stopping sets* and *codewords*. Recently, Karimi and Banihashemi proposed a characterization of LETSs, which was based on viewing a LETS as a layered superset (LSS) of a short cycle in the code's Tanner graph. However this characterization was not exhaustive. In this work, first, we complement LSS characterization by demonstrating how the remaining structures of LETSs can be characterized. Then, we propose a new characterization for LETSs of variable-regular LDPC codes. Compared to the LSS-based characterization, which is based on a single LSS expansion technique, the new characterization involves two additional expansion techniques. The introduction of the new techniques mitigates the search efficiency problem that LSS-based characterization/search suffers from. We prove that using the three expansion techniques, any LETS structure can be obtained starting from a simple cycle, no matter how large the size of the structure a or the number of its unsatisfied check nodes b are, i.e., the characterization is exhaustive. We also demonstrate that for the proposed characterization/search the length of the short cycles required to be enumerated is minimal. We also prove that the three expansion techniques, proposed here, are the only expansions needed for characterization of LETS structures starting from simple cycles in the graph.

Moreover, we generalize the proposed approach of variable-regular to irregular LDPC codes. We explain how the characterization of LETS structures in variable-regular graphs can be used to characterize the LETS structures of irregular graphs (in a given range of a and b values, exhaustively). This characterization corresponds to an exhaustive search algorithm. However, this approach is not applicable to irregular LDPC codes, if we are interested to find LETSs in a relatively wide range of

a and b. To overcome the above problem, we use a different approach to characterize/search LETSs of irregular codes. Also, we propose a graph based approach to find all ETSs (not necessarily LETSs) which are believed to be the error-prone structures of irregular LDPC codes.

In addition, we derive a lower bound on the size of the smallest elementary and non-elementary trapping sets for a given b in variable-regular LDPC codes. The derived lower bound demonstrates that the size of the smallest possible non-elementary trapping set is, in general, larger than that of an elementary trapping set with the same b value. This provides a theoretical justification as to why non-elementary trapping sets are often not among the most harmful trapping sets. Moreover, we propose an efficient search algorithm to provide an exhaustive/non-exhaustive list of NETSs in an interest range of a and b values. Moreover, we derive tight lower and upper bounds on the minimum distance d_{min} and stopping distance s_{min} of LDPC codes. The bounds, which are established using a combination of analytical results and search techniques, are applicable to both regular and irregular LDPC codes with a wide range of rates and block lengths. The search algorithms to find codewords and stopping sets are based on the ETSs and NETSs search algorithms provided in this thesis. Extensive simulation results on several LDPC codes demonstrate the accuracy and efficiency of the proposed algorithms. In particular, the algorithms are significantly faster than the existing search algorithms for finding the LETSs, ETSs, NETSs, stopping sets and codewords of LDPC codes.

To My Wife

Acknowledgments

First, I would like to thank my supervisor, Professor Amir H. Banihashemi for leading and encouraging me during my research. His guidance helped me in all the time of research and writing of this thesis.

Thanks must also go to my committee members for their helpful suggestions and discussions.

I can not express my appreciation to my wonderful wife, Ghazal who devoted her never ending love to me and was my partner towards success. I also appreciate the love and support of my family, specially my parents who are always the source inspiration and encouragement for me. And finally, many thanks to my friends, especially my friends in our research group.

Table of Contents

List of Tables

List of Figures

Nomenclature

Acronym	Description
AWGN	Additive White Gaussian Noise
BEC	Binary Erasure Channel
BSC	Binary Symmetric Channel
DPL	Dot Path Lollipop
EAS	Elementary Absorbing Set
ETS	Elementary Trapping Set
ETSL	Elementary Trapping Set with Leaf
FEAS	Fully Elementary Absorbing Set
FER	Frame Error Rate
LDPC	Low Density Parity Check
LETS	Leafless Elementary Trapping Set
LSS	Layered Superset
NETS	Non-Elementary Trapping Set
QC	Quasi-Cyclic
SNR	Signal to Noise Ratio
SS	Stopping Set
TS	Trapping Set

Chapter 1

Introduction

1.1 Motivation

Low-density parity-check (LDPC) codes, an interesting class of linear block codes, were first introduced by Gallager in 1962 [22] and rediscovered by Mackay and Neal in 1996 [56]. The main reason for the popularity of LDPC codes is their near Shannon limit performance with relatively low decoding complexity based on message-passing algorithms. These codes have been adopted in a number of standards such as IEEE 802.11n (WiFi) [100], IEEE 802.16e (WiMAX) [101], IEEE 802.22 (WRAN) [102], IEEE 802.3an (10GBASE-T Ethernet) [103], DVB-T2 [104] and DVB-S2 [105] and are best candidates in the next generation of optical networks [86], data storage systems [21] and 5G networks [4]. Message passing decoders are iterative algorithms that operate on Tanner graphs [82] of LDPC codes by computing messages at the nodes of the graph and passing them along the edges to the adjacent nodes. The two most well-known of such decoders are the sum-product (SP) and the min-sum (MS) decoders.

Finite-length LDPC codes under iterative decoding algorithms suffer from the *error floor* phenomenon. In the error floor region, the slope of error rate curves decreases when the channel quality improves beyond a certain point. In applications such as data storage and optical communication which require low bit error rates, it is critical to have a code with a low error floor. Due to the lack of comprehensive knowledge on the iterative message passing decoders, it is difficult to design practical LDPC codes with a guaranteed performance in the low error region. Moreover, estimating the performance of LDPC codes in such applications in the low bit error rate region is beyond the reach of Monte Carlo simulations. It is well-known that

error floor performance of LDPC codes is related to the presence of certain problematic graphical structures (error-prone structures) in the Tanner graph of the code, commonly referred to as *trapping sets* [66]. Although, the algorithms commonly used for the decoding of LDPC codes are iterative and suboptimal, the minimum distance still plays an important role in the performance of the iterative algorithms. This is, particularly, in relation to the so-called undetected errors. Moreover, minimum distance can be an upper bound on the size of the error-prone structures of the codes in the error floor region. The minimum distance of an LDPC code is characterized by the minimum weight of non-zero codewords, which can be themselves viewed as trapping sets in the code's Tanner graph. For these reasons, many studies have been focused on characterizing and finding the trapping sets.

Graphically, a trapping set (TS) is often seen as the induced subgraph of some variable nodes in the code's Tanner graph. In such a representation, the TS is commonly characterized by the number of its variable nodes a, and the number of its odd-degree (unsatisfied) check nodes b, and is said to belong to the class of (a, b) trapping sets. In binary erasure channel (BEC), the error-prone structures (EPSs) are called *stopping sets* [12], whose induced subgraphs contain no degree-1 check node. The weight of the smallest stopping set is called *stopping distance,* $s_{\min}$. Unlike the BEC, in other memoryless channels such as binary symmetric channel (BSC) and additive white Gaussian noise channel (AWGNC), the most harmful TSs are known to be *elementary trapping sets (ETSs)*, whose induced subgraphs contain only degree-1 and degree-2 check nodes. In particular, the *leafless* ETSs (LETSs), in which each variable node is connected to at least two even-degree (satisfied) check nodes, are recognized as the main culprit [95], [55], [62]. Also, a codeword of weight a can be viewed as a trapping set in the $(a, 0)$ class.

Exhaustive search of all the small EPSs in arbitrary, finite-length LDPC codes has been proven to be NP-hard [90]. In the past decade, some search algorithms have been proposed for finding the list of EPSs of the LDPC codes. However, all these algorithms suffer from two major problems. In many cases, the search algorithms are non-exhaustive, such as [1] for finding TSs, [97], [1], [88] for finding ETSs, [47] for finding LETSs, [39], [10] for finding low-weight codewords, and [68] for finding low-weight stopping sets. Even in the cases that the search algorithms are exhaustive, such as [97] for TSs, [23] for ETSs, [46] for LETSs, [53] for FASs, [49] for low-weight codewords, and [70] for low-weight stopping sets, they are highly limited to finding

small EPSs with small b values and/or LDPC codes with relatively short block lengths, small variable degrees and low-moderate rates.

The knowledge of trapping sets and their structure has been used in three applications: First, low error rates are beyond the reach of current Monte Carlo simulation methods. Due to the lack of general analytical results for determining the error floor of LDPC codes, the performance of a given LDPC code is often verified by simulation. The solution to this problem is to use importance sampling (IS) techniques which operate by biasing the noise toward the dominant trapping sets of the code [66]. This has attracted a great amount of interest in recent years in estimating the performance of LDPC codes in the error floor region. Also, more approaches have been proposed recently to estimate the error floor of LDPC codes [93], [92], [94], [77] [8], [16]. The input of all these approaches is the list of dominant trapping sets. Second, this knowledge has been used in modifying iterative decoders to have better error-floor performance [7], [27], [53], [45], [98]. Third, the knowledge of TSs has been used in constructing LDPC codes with low error floors [44], [2], [63], [51], [3], [14], [83]. Such works are based on eliminating the potentially dominant trapping sets.

In all the above applications, one needs the set of dominant TSs in the Tanner graph of the code. Due to the lack of comprehensive knowledge on the dominant trapping sets, it is necessary to provide an exhaustive list of TSs in a relatively wide range. However, attaining such a list, regardless of differences in the graphical structure of these sets and the sparsity of the underlying graph, is a hard problem [61]. The complexity of the existing search methods for finding problematic structures of size t in a code of length n becomes quickly intractable as n and t increase. On the other hand, in the next generation of communication systems, LDPC codes with large block lengths n, containing EPSs with relatively large size t, would be considered. It is therefore important to develop efficient algorithms for finding an exhaustive list of EPSs of LDPC codes.

1.2 Related Works

There has been a flurry of activity in characterization of trapping sets [88], [15], [13], [17], [74], [54], [63], [46], and in developing search algorithms to find them [70], [90], [97], [1], [91], [47], [53], [23]. Most of the works on TSs in the literature are non-exhaustive [1], [47], are limited to structured codes [96], [74], are concerned with

relatively small TSs [54], [97], only deal with a certain variable node degree [88], [63], or are applicable to relatively short block lengths [90], [53], [23].

In particular, in [1], using the modified impulse algorithm, the authors devised a technique to find a non-exhaustive list of trapping sets of a given short-length LDPC code. In [47], by examining the relationships between cycles and trapping sets, Karimi and Banihashemi proposed an efficient search algorithm to find the dominant LETSs of a given regular or irregular LDPC code. Although the algorithm proposed in [47] can find the LETSs with sufficiently large size, it provides no guarantee that the obtained list of LETSs is exhaustive. Laendner *et al.* [54] studied the characterization of small (a, b) trapping sets of size up to 8 ($a \leq 8$) and $b/a < 1$ in LDPC codes from Steiner Triple Systems (STS). STS LDPC codes are a special category of regular LDPC codes with variable node degree 3. In [88], [63], the authors presented the *trapping set ontology*, a database of trapping sets related by their topological structures, and used it to enumerate the dominant trapping sets and to construct structured LDPC codes free of small trapping sets. Specifically, by finding parent-child relationships between the trapping sets in regular LDPC codes with variable node degree 3, they proposed a search algorithm to enumerate these trapping sets. Due mainly to the tedious task of forming the trapping set ontology, the application of the technique of [88], [63] was limited to variable degree 3, and only a small range of trapping sets. Kyung and Wang [53] proposed an exhaustive search algorithm to enumerate the fully absorbing sets (FASs) of short regular and irregular LDPC codes. The algorithm is based on the branch-&-bound algorithm, a systematic enumeration of all candidate solutions, that is commonly used to solve NP-hard integer programming problems in the computer science literature [70]. The proposed algorithm, however, becomes quickly infeasible to use as the block length, n, and the size of the FASs, a, are increased. In general, the reach of the algorithm is limited to $n < 1000$ and $a < 7$, or $n < 1000$ and $a < 14$, if the number of unsatisfied check nodes is limited to $b < 3$. Also, all the reported variable-regular LDPC codes in [53] are limited to those with variable node degree 3. Recently, Falsafain and Mousavi [23], proposed a branch-&-bound algorithm to find the ETSs of LDPC codes. The presented 0-1 integer linear programming (ILP) formulation in [23] provided a tighter linear programming relaxation in comparison with the one in [53]. However, still the exhaustive search algorithm in [23] is only applicable to short-to-moderate length LDPC codes. In [97], the authors used the branch-&-bound approach to propose an exhaustive

search algorithm to find all the TSs of LDPC codes. However, similar to the other branch-&-bound algorithms, e.g., [53], this approach is only applicable to short block length codes (the block lengths of all the reported codes are less than 1008).

Recently, Karimi and Banihashemi [46] characterized the LETSs of variable-regular LDPC codes. They studied the graphical structure of the LETSs, and demonstrated that a vast majority of the non-isomorphic structures of LETSs are layered super sets (LSS) of short cycles, i.e., each structure $\mathcal{S}$ corresponds to a sequence of LETS structures that starts from a short cycle and grows one variable node at a time to reach $\mathcal{S}$. This characterization corresponds to a simple search algorithm that starts from the instances of short cycles in the Tanner graph and can find the LETSs with the LSS property. A shortcoming of this characterization/search, however, is that some of the LETS structures do not lend themselves to the above characterization, i.e., they are not LSSs of any of their cycle subsets. Such structures are labeled as "NA" in [46].

The research on the minimum distance of LDPC codes has followed two general directions. In one direction, analytical lower and upper bounds on d_{min} have been derived [82], [79], [25], [75], [6]. In [82], using a graph-based analysis, lower bounds on the minimum distance of variable-regular LDPC codes were obtained. These bounds were then simplified in [89] for a given girth and variable degree. Furthermore, in [25], [75], [6], analytical upper bounds on the minimum distance of quasi cyclic (QC) LDPC codes were derived. In another direction, search algorithms have been proposed to find d_{min} of LDPC codes, or to provide lower and/or upper bounds on d_{min} [41], [68], [10], [49], [39], [50], [6]. Error impulse method, originally proposed for turbo codes, was used in [41], and later modified in [10], to estimate d_{min} of LDPC codes. This approach, called Nearest Non-zero Codeword Search (NNCS), is based on adding a random and/or deterministic noise to the all-zero codeword and obtaining the nearest non-zero codewords as a result of decoding. In [39], the authors proposed a probabilistic method based on the algorithm of [78] to find upper bounds on d_{min} of LDPC codes. This algorithm is applicable only to LDPC codes with relatively small block length ($n \leq 1000$) and relatively small minimum distance ($d_{min} \leq 20$). Based on two integer programming models, Keha and Duman [49] proposed a branch and cut algorithm to find the minimum distance of an LDPC code. If the algorithm of [49] fails in finding d_{min}, it provides a lower and an upper bound on d_{min}. The algorithm, however, becomes quickly infeasible to use (in terms of time or memory)

as the block length, n, the rate, R, and/or d_{min} are increased. As a result, in [49], only the minimum distances of LDPC codes with moderate rate ($R = 0.5$) and relatively short block length ($n = 512$) are reported. In [50], the authors studied the topological structure of codewords of weight a with $a \leq 14$, in regular LDPC codes with variable degree 3 and girth 8, and proposed a search algorithm to find such low-weight codewords. The algorithm is, however, limited to only regular LDPC codes with variable degree 3 and girth 8. In summary, most of the search algorithms provided in the literature result in estimates of d_{min} that are in fact upper bounds. These algorithms are also mainly limited to short- and moderate-length LDPC codes.

It is well-known that, in general, finding the stopping distance, $s_{\min}$, for an arbitrary Tanner graph is an NP-hard problem [52]. However, there exist some search algorithms in the literature to find the list of small stopping sets (or just $s_{\min}$) for LDPC codes mostly with short-moderate block lengths and/or low-moderate rates and/or small variable degrees (for variable-regular ones) [68], [42], [40], [90], [70], [71], [72], [19], [6]. In [90], the authors proposed a branch-&-bound search algorithm to find small stopping sets. The proposed algorithm, however, becomes quickly infeasible to use as the block length, n, and stopping distance $s_{\min}$ are increased. Also, all the three LDPC codes studied in this paper are structured regular codes with variable degree 3 and rate less than or equal to 0.5. Using the Stern's probabilistic algorithm [78], the authors in [42], [40] proposed search algorithms for computing the minimum size of stopping sets of LDPC codes. However, the proposed approaches are applicable to only short block length random codes or medium block length structured ones. Also, all the LDPC codes studied in [42], [40] are variable-regular with variable degree 3 and rate less than or equal to 0.5. Authors in [70] and [71] proposed branch-&-bound algorithms to find all the stopping sets of LDPC codes. To the best of our knowledge, this is the most efficient exhaustive search of stopping sets available in the literature. However, similar to all other branch-&-bound algorithms, this approach is highly limited to the short block lengths and thus, determining the minimum size of stopping sets of large block length LDPC codes would be a difficult task which requires exponentially growing complexity. For example, except two random regular codes with rate 0.5 and block length of 504, all the studied codes in [70] and [71] are structured codes[1] with length less than or equal to 4896. Also, all the regular codes studied in [70] and [71] are codes with variable degree 3 and rate 0.5.

[1]The structural properties of the codes are used in [70] and [71] to speed up the algorithms.

1.3 Summary of Contributions

In this thesis, we address the above challenges as described in the following. Our main contributions in this thesis, which are published or submitted for publication in [28], [29], [30], [31], [32], [33], [34], [35], [36], [37] can be summarized as follows:

- In [28] and [29], we complement the results of [46]. By careful examination of NA structures, we demonstrate that they are all LSSs of some basic graphical structures which are slightly more complex than cycles.

- We propose a novel hierarchical graph-based expansion approach to characterize leafless elementary trapping sets (LETS) of variable-regular LDPC codes [30], [31], [32]. The new characterization, allows us to devise search algorithms that are provably efficient in finding all the instances of (a, b) LETS structures with $a \leq a_{max}$ and $b \leq b_{max}$, for any choice of a_{max} and b_{max}, in a guaranteed fashion.

- We propose a novel hierarchical graph-based expansion approach to characterize elementary trapping sets (ETS) of irregular LDPC codes [33], [37]. The proposed characterization corresponds to an efficient search algorithm to find all the ETS structures with $a \leq a_{max}$ and $b \leq b_{max}$, for any choice of a_{max} and b_{max}, in a guaranteed fashion.

- We derive a lower bound on the size of the smallest possible elementary and non-elementary trapping sets (NETS) with a given b in variable-regular LDPC codes [35]. Also, we propose a graph-based approach to characterize and search the NETSs of variable-regular LDPC codes [36].

- We obtain lower and upper bounds on the minimum distance d_{min} of LDPC codes [34]. For the lower bound, by performing a graph-based search algorithm, we either obtain an elementary codeword with the smallest weight d_{min}, or establish a lower bound on d_{min}. For the upper bound, we modify our search algorithm to reach elementary codewords of larger weights at the cost of being non-exhaustive.

- We obtain lower and upper bounds on the stopping distance of LDPC codes [36]. For the lower bound, by performing an exhaustive search algorithm, we either obtain a stopping set with the smallest weight s_{min}, or establish a lower bound

on s_{min}. For the upper bound, we modify our search algorithm to reach stopping sets of larger weights at the cost of being non-exhaustive.

Recently, Karimi and Banihashemi demonstrated that a large majority of the LETS structures of variable-regular LDPC codes are layered supersets (LSS) of short cycles. We complement this characterization by demonstrating that the remaining structures of LETSs, that are not LSS of short cycles, are all LSS of a small number of other graphical structures within the Tanner graph of the code. This together with the results of Karimi and Banihashemi provides a simple LSS search algorithm that can find all the (a, b) LETSs of any variable-regular LDPC code for any size a and any number of unsatisfied check nodes b in a guaranteed fashion. Although, the search algorithm itself is simple, one may need to enumerate basic structures with large size in the Tanner graph of the code, as the input to the search algorithm. The multiplicity of such structures increases rapidly with the size and thus the enumeration, storage and processing of these structures pose a practical hurdle in implementing the search algorithm.

Motivated by this, we propose a novel hierarchical graph-based expansion approach to characterize leafless elementary trapping sets (LETS) of variable-regular LDPC codes. The proposed characterization is based on three basic expansion techniques, dubbed, *dot*, *path* and *lollipop*. Each LETS structure $\mathcal{S}$ is characterized as a sequence of embedded LETS structures that starts from a simple cycle, and grows in each step by using one of the three expansions, until it reaches $\mathcal{S}$. It is demonstrated that the new characterization, called *dpl*, is minimal, in that, none of the expansions in the sequence can be divided into smaller expansions and still maintain the property that all the embedded sub-structures are LETSs. Moreover, it is proved that any minimal characterization based on embedded LETS structures must only use one of the three expansions, *dot*, *path* and *lollipop*, at each step. The minimality of the characterization, allows us to devise search algorithms that are provably efficient in finding all the instances of (a, b) LETS structures with $a \leq a_{max}$ and $b \leq b_{max}$, for any choice of a_{max} and b_{max}, in a guaranteed fashion. The efficiency of the search is a consequence of the fact that to enumerate the instances of larger LETS structures, the proposed algorithm searches for instances of smaller LETS structures that are of interest themselves, i.e., their a and b values are in the range of interest.

In irregular Tanner graphs, for a given class of LETSs, the variety of non-isomorphic LETS structures would increase significantly compared to that of variable-regular Tanner graphs. However, by discussing the similarities and differences of variable-regular and irregular Tanner graphs, we explain how *dpl* characterization/search can be used to search the LETSs of irregular graphs exhaustively. Also, based on literature and our experimental results, ETSs are the problematic structures of irregular LDPC codes. Therefore, in addition to the LETSs, we characterize the ETSs which are not LETSs and provide an efficient search algorithm to exhaustively find them, starting from the LETSs in the interest range.

Empirical results have shown that often the most harmful trapping sets are elementary. We derive a lower bound on the size of the smallest non-elementary trapping sets for a given b in variable-regular LDPC codes. The derived lower bound demonstrates that the size of the smallest possible non-elementary trapping set is, in general, larger than that of an elementary trapping set with the same b value. This provides a theoretical justification as to why non-elementary trapping sets are often not among the most harmful trapping sets. Also, we propose an efficient exhaustive search algorithm to find NETSs in variable-regular graphs. This along with the theoretical justification support that, potentially, in the harmful classes of TSs, there is no NETSs.

Also, we obtain lower and upper bounds on the minimum distance d_{min} of LDPC codes. The bounds are derived by categorizing the non-zero codewords of an LDPC code into two categories of elementary and non-elementary. The first category contains codewords whose induced subgraph has only degree-2 check nodes. We use the *dpl* search algorithm to find the elementary codewords of an LDPC code with weight less than a certain value a_{max}, exhaustively. We also derive a lower bound L_{ne} on the weight of non-elementary codewords. By performing the search with $a_{max} = L_{ne}$, we either obtain an elementary codeword with the smallest weight d_{min}, or establish the lower bound of L_{ne} on d_{min}. For the upper bound, we modify our search algorithm to reach elementary codewords of larger weights at the cost of being non-exhaustive. Once such a codeword is found, its weight acts as an upper bound on d_{min}. In general, if we succeed in finding the exact minimum distance of an LDPC code, we do so in much higher speed than existing algorithms. If we fail, and establish the lower bound of $L_{ne} \leq d_{\min}$, this lower bound in many cases is tight. Also, in the latter case, our algorithm for finding an upper bound on $d_{\min}$ is much faster than the algorithms in

the literature.

Moreover, we obtain lower and upper bounds on the stopping distance s_{min} of LDPC codes. By selecting a proper $a_{\max} = L_{ss}$, we use the exhaustive LETS and NETS search algorithms to find the lowest weight stopping sets with weight less than L_{ss}. If successful, we have in fact found $s_{\min}$. If the algorithm fails in finding a stopping set of weight less than L_{ss}, then, we establish the lower bound of $s_{\min} \geq L_{ss}$ on $s_{\min}$. In the latter case, to find an estimate (upper bound) for $s_{\min}$, we modify the LETS and NETS search algorithms, to extend the range of search ($a_{\max}$), at the cost of sacrificing the exhaustiveness of the algorithm. In general, if we succeed in finding the exact stopping distance of an LDPC code, we do so in much higher speed than existing algorithms. If we fail, and establish the lower bound of $L_{ss} \leq s_{\min}$, this lower bound in many cases is tight. Also, in the latter case, our algorithm for finding an upper bound on $s_{\min}$ is often much faster than the algorithms in the literature.

In this thesis, we have studied *all the EPSs* of LDPC codes. These are *ETSs, LETSs, NETSs, codewords* and *stopping sets*. We have characterized these EPSs and proposed efficient exhaustive search algorithms for finding them. The proposed algorithms are applicable to regular and irregular codes with any given girth, block length, rate and degree distribution. The base of all the proposed algorithms is the characterization of LETSs in variable-regular graphs. It is a hierarchical graph-based expansion approach and each structure is characterized as a sequence of embedded structures that starts from a short simple cycle. Therefore, the initial input of all the proposed search algorithms is a list of short simple cycles in the Tanner graph of the code. Recently in [9], it has been shown that for a random bipartite graph, with a given degree distribution, the distributions of cycles of different length tend to independent Poisson distributions, as the size of the graph tends to infinity. Therefore, by increasing the block length of the codes, the multiplicity of the simple cycles are independent of the block length. This implies that, after finding the list of short simple cycles, the proposed searches in this thesis are rather independent of the block length. On the other hand, it is easy to show that the complexity of searching the simple cycles increases linearly with the block length of the code. This low dependency to the block length is the main advantage of the proposed search algorithm in comparison to the ones in the literature. The proposed *dpl*-based algorithms in this work are the most efficient exhaustive search algorithms available for finding ETSs, LETSs, NETSs, codewords and stopping sets of LDPC codes. It is also the most general,

in that, it is applicable to codes with any degree distribution, girth, rate and block length.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows. Basic definitions, notations and background are provided in Chapter 2. In Chapter 3, we complement the results of LSS-based search algorithm [46]. In Chapter 4, we revisit the LSS-based approach and discuss its shortcomings. In Chapter 5, we study the graphical structures of leafless elementary trapping sets in variable-regular LDPC codes and propose the *dpl* approach to find them exhaustively. In Chapter 6, we study the graphical structures of elementary trapping sets in irregular LDPC codes, and devise an efficient *dpl*-based algorithm to find them exhaustively. In Chapter 7, we derive a lower bound on the size of the smallest possible elementary and non-elementary trapping sets. In Chapter 8, we obtain lower and upper bounds on the minimum distance d_{min} of LDPC codes. In Chapter 9, we characterize and devise search algorithms for non-elementary trapping sets in variable-regular LDPC codes. In Chapter 10, we obtain lower and upper bounds on the stopping distance of LDPC codes. Conclusion and future works are presented in Chapter 11.

Chapter 2

Preliminaries

2.1 Graph Theory

Consider an undirected graph $G = (F, E)$, where the two sets $F = \{f_1, \ldots, f_k\}$ and $E = \{e_1, \ldots, e_m\}$, are the sets of *nodes* and *edges* of G, respectively. We say that an edge e is *incident* to a node f if e is connected to f. If there exists an edge e_k which is incident to two distinct nodes f_i and f_j, we call these nodes *adjacent*. In this case, we represent e_k by $f_i f_j$ or $f_j f_i$. The *neighborhood* of a node f, denoted by $\mathcal{N}(f)$, is the set of nodes adjacent to f. The degree of a node f is denoted by *deg(f)*, and is defined as the number of edges incident to f. The *maximum degree* and the *minimum degree* of a graph G, denoted by $\Delta(G)$ and $\delta(G)$, respectively, are defined to be the maximum and minimum degree of its nodes, respectively.

Given an undirected graph $G = (F, E)$, a *walk* between two nodes f_1 and f_{k+1} is a sequence of nodes and edges $f_1,\ e_1,\ f_2,\ e_2,\ \ldots,\ f_k,\ e_k,\ f_{k+1}$, where $e_i = f_i f_{i+1}$, $\forall i \in [1, k]$. In this definition, the nodes $f_1, f_2, \ldots, f_{k+1}$ are not necessarily distinct. The same applies to the edges $e_1, e_2, \ldots, e_k$. A *path* is a walk with no repeated nodes or edges, except the first and the last nodes that can be the same. If the first and the last nodes are distinct, we call the path an *open path*. Otherwise, we call it a *cycle*. The *length* of a walk, a path, or a cycle is the number of its edges. A *lollipop walk* is a walk $f_1,\ e_1,\ f_2,\ e_2,\ \ldots,\ f_k,\ e_k,\ f_{k+1}$, such that all the edges and all the nodes are distinct, except that $f_{k+1} = f_m$, for some $m \in (1, k)$. A *chord* of a cycle is an edge which is not part of the cycle but is incident to two distinct nodes in the cycle. A *simple cycle* or a *chordless cycle* is a cycle which does not have any chord. Throughout this work, we use the notation s_k for a simple cycle of length k. Notation c_k is used for cycles of length k that do have at least one chord.

A graph is called *connected* when there is a *path* between every pair of nodes. A *tree* is a connected graph that contains no cycles. A *rooted tree* is a tree in which one specific node is assigned as the *root*. The *depth of a node* in a rooted tree is the length of the path from the node to the root. The *depth of a tree* is the maximum depth of any node in the tree. *Depth-one tree* is a tree with depth one. A node f is called *leaf* if $deg(f) = 1$. Although this terminology is commonly used for trees, in this work, we use it for a general graph that may contain cycles. A *leafless graph* is a connected graph G with $\delta(G) \geq 2$.

The graphs $G_1 = (F_1, E_1)$ and $G_2 = (F_2, E_2)$ are *isomorphic* if there is a bijection $p : F_1 \to F_2$ such that nodes $f_1, f_2 \in F_1$ are joined by an edge if and only if $p(f_1)$ and $p(f_2)$ are joined by an edge. Otherwise, the graphs are *non-isomorphic*.

2.2 LDPC Codes

A graph $G = (F, E)$ is called *bipartite* if the set F can be partitioned into two disjoint subsets V and C ($F = V \cup C$ and $V \cap C = \emptyset$). Any $m \times n$ parity check matrix H of a binary LDPC code $\mathcal{C}$ can be represented by its bipartite Tanner graph $G = (V \cup C, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of variable nodes and $C = \{c_1, c_2, \ldots, c_m\}$ is the set of check nodes. The degrees of nodes v_i and c_i are denoted by d_{v_i} and d_{c_i}, respectively. A Tanner graph is called *variable-regular* with variable degree d_{v} if $d_{v_i} = d_{\mathrm{v}}$, $\forall\ v_i \in V$. A $(d_{\mathrm{v}}, d_{\mathrm{c}})$-regular Tanner graph is a variable-regular graph in which $d_{c_i} = d_{\mathrm{c}}$, $\forall\ c_i \in C$. A Tanner graph is called *irregular* if it has multiple variable and check node degrees. An irregular LDPC code is usually described by its variable and check node degree distributions, $\lambda(x) = \sum_{i=d_{\mathrm{v_{min}}}}^{d_{\mathrm{v_{max}}}} \lambda_i x^{i-1}$ and $\rho(x) = \sum_{i=d_{\mathrm{c_{min}}}}^{d_{\mathrm{c_{max}}}} \rho_i x^{i-1}$, respectively, where λ_i and ρ_i are the fractions of edges in the Tanner graph that are incident to a degree-i variable and check nodes, respectively. The terms $d_{\mathrm{v_{max}}}, d_{\mathrm{v_{min}}}$ are the maximum and minimum degrees of variable nodes, respectively and the terms $d_{\mathrm{c_{max}}}, d_{\mathrm{c_{min}}}$ are the maximum and minimum degrees of check nodes, respectively in the Tanner graph. For a subset $\mathcal{S}$ of V, the subset $\Gamma(\mathcal{S})$ of C denotes the set of neighbors of $\mathcal{S}$ in G. The *induced subgraph* of $\mathcal{S}$ in G, denoted by $G(\mathcal{S})$, is the graph with the set of nodes $\mathcal{S} \cup \Gamma(\mathcal{S})$ and the set of edges $\{f_i f_j \in E : f_i \in \mathcal{S}, f_j \in \Gamma(\mathcal{S})\}$. The set of check nodes with odd and even degrees in $G(\mathcal{S})$ are denoted by $\Gamma_o(\mathcal{S})$ and $\Gamma_e(\mathcal{S})$, respectively. The terms *unsatisfied check nodes* and *satisfied check nodes* are used to refer to the check nodes in $\Gamma_o(\mathcal{S})$ and $\Gamma_e(\mathcal{S})$, respectively. The *size* of an induced

subgraph $G(\mathcal{S})$ is defined to be the number of its variable nodes. We assume that an induced subgraph is connected. Disconnected subgraphs can be considered as the union of connected ones. The length of the shortest cycle in a Tanner graph is called *girth*, and is denoted by g. We study the variable-regular graphs with $d_{\mathrm{v}} \geq 3$, that are free of 4-cycles ($g > 4$). All the induced subgraphs with the same size a, and the same number of unsatisfied check nodes b, are considered to belong to the same (a, b) *class*.

Given a Tanner graph G, a set $\mathcal{S} \subset V$ is called an *(a,b) trapping set (TS)* if $|\mathcal{S}| = a$ and $|\Gamma_o(\mathcal{S})| = b$. Alternatively, $\mathcal{S}$ is said to belong to the *class of (a,b) TSs*. Parameter a is referred to as the *size* of the TS. In this thesis, depending on the context, the term "trapping set" may be used to refer to the set of variable nodes $\mathcal{S}$, or to the induced subgraph $G(\mathcal{S})$ of $\mathcal{S}$ in the Tanner graph G. Similarly, we may use $\mathcal{S}$ to mean $G(\mathcal{S})$. An *elementary trapping set (ETS)* is a trapping set for which all the check nodes in $G(\mathcal{S})$ have degree 1 or 2. A set $\mathcal{S} \subset V$ is called an *(a,b) absorbing set (AS)* if $\mathcal{S}$ is an (a, b) trapping set and all the variable nodes in $\mathcal{S}$ are connected to more nodes in $\Gamma_e(\mathcal{S})$ than in $\Gamma_o(\mathcal{S})$. An *elementary absorbing set (EAS)* $\mathcal{S}$ is an absorbing set for which all the check nodes in $G(\mathcal{S})$ have degree 1 or 2. A *fully absorbing set (FAS)* $\mathcal{S} \subset V$ is an absorbing set for which all the nodes in $V \backslash \mathcal{S}$ have strictly more neighbors in $C \backslash \Gamma_o(\mathcal{S})$ than in $\Gamma_o(\mathcal{S})$. A set $\mathcal{S} \subset V$ is called an *(a,b) fully elementary absorbing set (FEAS)* if $\mathcal{S}$ is an (a, b) EAS and if all the nodes in $V \backslash \mathcal{S}$ have strictly more neighbors in $C \backslash \Gamma_o(\mathcal{S})$ than in $\Gamma_o(\mathcal{S})$. A *non-elementary trapping set (NETS)* is a trapping set which is not elementary.

The following lemma shows that for variable-regular LDPC codes, depending on d_{v} being odd or even, some classes of trapping sets cannot exist.

Lemma 1. *In a variable-regular Tanner graph with variable degree d_{v}, (a) if d_{v} is odd, then there does not exist any (a, b) TS with odd a and even b, or with even a and odd b; and (b) if d_{v} is even, then there does not exist any (a, b) TS with odd b.*

Proof. We first prove the first part of Claim (a), where a is odd and b is even. The proof of the second part is similar. We note that the number of edges in the subgraph $G(\mathcal{S})$ of the trapping set $\mathcal{S}$ under consideration is ad_{v}. For the case where d_{v} and a are both odd numbers, the number of edges in $G(\mathcal{S})$ will be an odd number. Now, counting the number of edges incident to the check nodes of $G(\mathcal{S})$, we note that the contribution of satisfied check nodes is always an even number. If the number of unsatisfied check nodes b is even, then the contribution of unsatisfied check nodes

will also be an even number, implying that the number of edges incident to check nodes is an even number. This is a contradiction.

For Claim (b), we note that since d_v is even, the number of edges of $G(\mathcal{S})$, counted from the variable side of the Tanner graph, i.e., ad_v, is an even number. From the check side, however, if b is an odd number, the contribution of unsatisfied check nodes will be an odd number. This added to the even contribution of satisfied check nodes will result in an odd number. Again, a contradiction. ■

2.3 Graph Representations of Trapping Sets

Elementary trapping sets are the subject of this thesis. To simplify the representation of ETSs, similar to [46], we use an alternate graph representation of ETSs, called *normal graph*. The normal graph of an ETS $\mathcal{S}$ is obtained from $G(\mathcal{S})$ by removing all the check nodes of degree one and their incident edges, and by replacing all the degree-2 check nodes and their two incident edges by a single edge. It is easy to see that there is a one-to-one correspondence between the bipartite graph $G(\mathcal{S})$ and the normal graph of $\mathcal{S}$ for variable-regular LDPC codes. However this correspondence does not exist for irregular LDPC codes.

Lemma 2. *Consider the normal graph of an (a, b) ETS structure of a variable-regular Tanner graph with variable degree d_v. The number of nodes and edges of this normal graph are a and $(ad_v - b)/2$, respectively. We thus have $b = ad_v - 2e$, where e is the number of edges of the normal graph.*

We call a set $\mathcal{S} \subset V$ an *(a,b) leafless ETS (LETS)* if $\mathcal{S}$ is an (a, b) ETS and if the normal graph of $\mathcal{S}$ is leafless.

Example 1. *Fig. 2.1(a) represents a LETS in the $(4, 2)$ class in a variabl-regular Tanner graph with $d_v = 3$ and its leafless normal graph, while Fig. 2.1(b) represents an ETS in the $(4, 4)$ class and its normal graph with a leaf. Symbols □ and ■ are used to represent satisfied and unsatisfied check nodes in the induced bipartite subgraphs, respectively, and the symbol ○ is used to represent variable nodes in both the induced subgraphs and normal graphs.*

Unlike the variable-regular Tanner graphs, there is not a one-to-one correspondence between an ETS and its normal graph for irregular graphs. In other words,

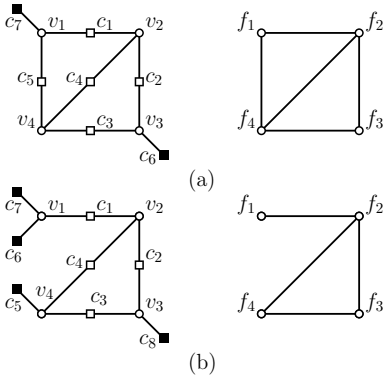


Figure 2.1: (a) A LETS in the $(4, 2)$ class and its leafless normal graph, (b) An ETS in the $(4, 4)$ class and its normal graph which has a leaf (f_1).

for an irregular Tanner graph, the number of edges in the normal graph representation of an ETS is not uniquely mapped to the number of unsatisfied check nodes in the ETS. To have a one-to-one correspondence, in addition to the normal graph, the extra information about the degrees of variable nodes involved in the ETS is also required. For this, we introduce a new graphical representation of an ETS, which we call *quasi-normal* representation. The *quasi-normal graph* of an ETS $\mathcal{S}$ is obtained from $G(\mathcal{S})$ by replacing all the check nodes (degree-one or two) and their incident edges by a single edge. In this representation, the edges that are connected to only one node (singly-connected edges) are responsible for preserving the degree of variable nodes. It is easy to see that there is a one-to-one correspondence between $G(\mathcal{S})$ and the quasi-normal graph of $\mathcal{S}$ for any regular or irregular LDPC code. We continue to use the normal graph representation for irregular graphs. Such a representation can be considered as the image or projection of quasi-normal graphs into the space of normal graphs, where such a projection involves dropping all the singly-connected edges. In general, in an irregular Tanner graph, multiple ETS structures with different quasi-normal graphs may have the same normal graph representation. We also continue to use the same definition of LETS for irregular graphs, i.e., an ETS $\mathcal{S}$ is LETS if the normal graph of $\mathcal{S}$ is leafless.

Example 2. *Fig. 2.2 shows the induced subgraphs of four LETS structures in an irregular Tanner graph with variable node degrees 3 and 4. The figure also includes the corresponding quasi-normal graphs and the normal graph representation of the four structures. It can be seen that all four non-isomorphic LETS structures have the*

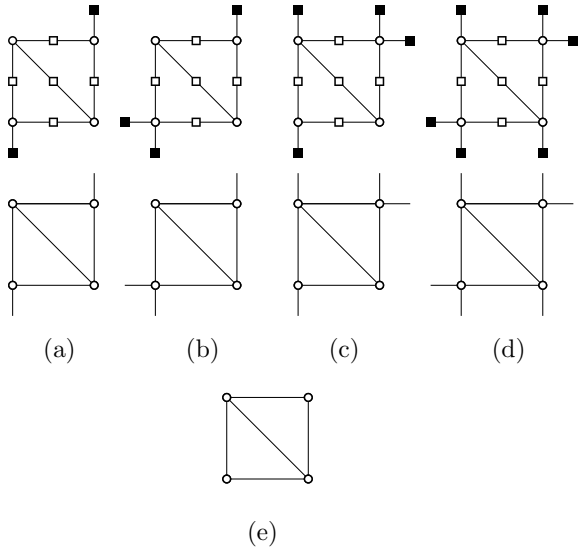


Figure 2.2: Tanner graph and quasi-normal graph representations of LETSs in an irregular Tanner graph with variable degrees 3 and 4 in (a) $(4, 2)$, (b) $(4, 3)$, (c) $(4, 4)$, (d) $(4, 6)$ classes and (c) their normal graph.

same normal graph.

In the following, when we are concerned with the characterization of LETS structures, we use the quasi-normal/normal graph representation. On the other hand, when we discuss search algorithms, since the search is performed on a Tanner graph, we are concerned with the bipartite graph representation of a LETS structure. Nevertheless, for consistency and to prevent confusion, even in the context of search algorithms, we still use terminologies corresponding to normal graphs. For example, we use "all the instances of s_k" to mean "all the instances of the structure whose normal graph is s_k."

2.4 Nauty Program

Nauty [60], is a package of programs for computing automorphism groups of graphs. There is a program called *geng* in this package that generates all the non-isomorphic graphical structures with a given number of nodes and edges very efficiently. This program has many input options to generate connected graphs, triangle-free graphs, or graphs with certain minimum and maximum node degrees. In this work, we use *geng* to generate all the non-isomorphic graphical structures of LETSs in different

classes. To simplify this task, we use the normal graph representation of LETSs. Since we are interested in connected LETS structures in variable-regular Tanner graphs with variable degree d_{v} that are 4-cycle free, we look for normal graphs with no parallel edges and with minimum and maximum node degrees equal to $\delta(G) \geq 2$ and $\Delta(G) \leq d_{\mathrm{v}}$, respectively. By setting the range of node degrees to $[2, d_{\mathrm{v}}]$, the number of nodes to a, the number of edges to $(ad_{\mathrm{v}} - b)/2$, and by choosing the simple connected graph in *geng*'s input options, this program generates all the non-isomorphic graphical structures of connected LETSs in the (a, b) class for variable-regular Tanner graphs with variable degree d_{v}. If we are interested in Tanner graphs with $g \geq 8$, we also add the option of "triangle-free."

Chapter 3

LSS based Characterization/Search of LETSs of Variable-Regular LDPC Codes

3.1 Introduction

In this chapter, we complement the results of [46], by careful examination of NA structures, we demonstrate that they are all LSSs of some basic graphical structures which are only slightly more complex than cycles.

3.2 LSS based Characterization

Recently, Karimi and Banihashemi [46] studied the graphical structure of LETSs for variable-regular LDPC codes, and demonstrated that a vast majority of the non-isomorphic structures of LETSs are *layered supersets (LSS)* of short cycles, i.e., each structure $\mathcal{S}$ corresponds to a sequence of LETS structures that starts from a short cycle and grows one variable node at a time to reach $\mathcal{S}$. In particular, for variable-regular Tanner graphs with variable degrees $d_{\rm v} = 3, 4, 5$ and 6, and girths $g = 6$ and 8, tables are provided in [46] that show the LSS properties of different non-isomorphic structures for different (a, b) classes of LETSs, for $a, b < 10$, where a is the size of the LETS and b is the number of odd-degree (unsatisfied) check nodes in the subgraph induced by the trapping set. Probably, the most significant consequence of this characterization is that it corresponds to a simple search algorithm that starts from the short cycles of the graph and can find all the LETSs with the LSS property in a guaranteed fashion. The shortcoming, however, is that some of the (a, b) LETS structures (within the range $a, b < 10$) do not lend themselves to the above characterization,

i.e., they are not LSSs of any of their cycle subsets. These structures are labeled as "NA" in [46]. For example, for variable-regular LDPC codes with $d_v = 4$ and $g = 6$, there are a total of 178 non-isomorphic NA structures within the range $a, b < 10$. This implies that these structures cannot be found efficiently by using the LSS-based search algorithm of [46] if one initiates the search by a set of cycles in the graph (no matter how large the maximum length of the cycles in the set is).

3.3 LSS-Prime LETS Structures

We call these basic structures *prime* with respect to the LSS property, or "LSS-prime" in brief, implying that they are not layered supersets of smaller LETSs. (By this definition, we include cycles that are not LSS of any of their subsets in the set of LSS-prime structures.) We show that for variable degrees $d_v = 3, 4, 5, 6$, girths $g = 6, 8$, and (a, b) classes with $a, b < 10$, the non-cycle LSS-prime structures are all composed of two short cycles, and they all fit into one of the following two categories: (i) The two cycles share either a variable node or a number of variable and check nodes (and their connecting edges), (ii) the two cycles are connected by a chain (of interleaving check and variable nodes).[1] For example, for the case of $d_v = 4$ and $g = 6$, all the 178 NA structures appear to be LSSs of only 7 non-cycle LSS-prime structures. An important consequence of this discovery is that, by including the non-cycle LSS-prime structures of a Tanner graph in the initial set (along with short cycles), the simple LSS-based search algorithm can find all the (a, b) LETSs (with $a, b < 10$) of the graph in a guaranteed fashion.[2]

We note that if a LETS $\mathcal{S}$ of size a is an LSS of one of its elementary trapping subsets $\mathcal{S}'$ of size a', then $\mathcal{S}$ can be efficiently found in a Tanner graph starting from $\mathcal{S}'$ using $a - a'$ successive applications of the following LSS-based search algorithm: add to the input LETS $\mathcal{S}_{\text{in}}$ any variable node that has at least two connections to the set of degree-1 check nodes in $\mathcal{S}_{\text{in}}$ and no connection to degree-2 check nodes in $\mathcal{S}_{\text{in}}$.

For variable-regular graphs with $d_v = 3, 4, 5$, and $g = 6$, the multiplicity of non-isomorphic structures of (a, b) LETSs and EASs which are reported as "NA" in [46]

[1] The constraints imposed on d_v, g as well as on a and b values are to follow those of [46]. Our results can be extended to larger values for these parameters. Covering larger values of a and b, however, requires a larger number of more complex non-cycle LSS-prime structures.

[2] The guaranteed property or exhaustiveness of the search is a direct consequence of the fact that all non-isomorphic structures for every (a, b) class of LETSs are examined for their LSS properties, as described above.

Table 3.1: multiplicity of non-isomorphic structures for different (a, b) LETS and EAS classes of variable-regular graphs with $d_v = 3, 4, 5$ and $g = 6$, which are reported as "NA" in [46]

d_v	3	4		5	
(a, b)	LETS	LETS	EAS	LETS	EAS
	(6,4): 1	(5,8): 1	-	(8,8): 1	-
	(7,3): 1	(6,8): 1	-	(9,7): 3	-
	(7,5): 2	(7,6): 1	-	(9,9): 19	(9,9): 1
	(8,2): 2	(7,8): 5	-		
	(8,4): 5	(8,4): 1	-		
	(8,6): 5	(8,6): 7	(8,6): 1		
	(9,1): 1	(8,8): 20	-		
	(9,3): 10	(9,4): 5	(9,4): 2		
	(9,5): 17	(9,6): 33	(9,6): 3		
	(9,7): 7	(9,8): 104	(9,8): 2		
Total	51	178	8	23	1

are listed in Table 3.1. (Note that for $d_v = 3$, LETSs are the same as EASs.)

There is no NA structure for variable-regular graphs with $d_v = 6$ and $g = 6$ or with $d_v = 4, 5, 6$ and $g = 8$ in the range of $a, b < 10$. For variable-regular graphs with $d_v = 3$ and $g = 8$, based on Table II of [46], there are only 2, 1 and 3 non-isomorphic NA structures for $(8, 6)$, $(9, 5)$ and $(9, 7)$ LETS classes, respectively. These structures are clearly a subset of NA structures for graphs with $d_v = 3$ and $g = 6$.

Example 1: In Fig. 3.1, we have shown the normal graph representation of all the possible non-isomorphic structures of the $(8, 6)$ ETS class for variable-regular graphs with $d_v = 3$ and $g = 8$. One can see that the two structures in Figs. 3.1(a),(b) are LSSs of a cycle of length $g + 6 = 14$. The structures in Figs. 3.1(c),(d) are cycles of length $g + 8 = 16$, and thus by definition, are LSSs of a cycle of length 16. In fact, Figs. 3.1(c),(d) are examples of LSS-prime structures that are cycles. Structures in Figs. 3.1(e),(f), however, are neither LSSs of a cycle nor LSS-prime cycles. They are both examples of non-cycle LSS-prime structures.

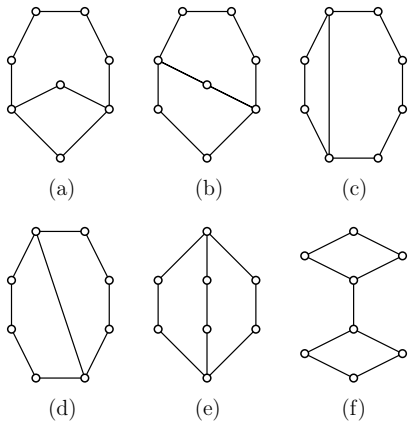


Figure 3.1: Normal graphs of all possible non-isomorphic structures of the class of $(8, 6)$ LETSs within variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 8$.

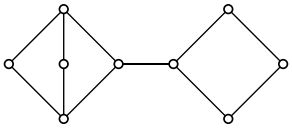


Figure 3.2: Normal graph of the only structure of the class of $(9, 5)$ LETSs within variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 8$.

Example 2: Fig. 3.2 shows the only NA structure of $(9, 5)$ LETS class for variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 8$. It is easy to see that this structure is an LSS of the $(8, 6)$ LETS structure shown in Fig. 3.1(f). The structure of Fig. 3.2 is thus not LSS-prime. It can however be found efficiently in a Tanner graph by a one-level expansion of the LSS-prime structure of Fig. 3.1(f) using the LSS-based search algorithm.

We carefully examine the graphical structure of all the LETS structures that are not LSSs of any cycles and are thus identified by "NA" in [46]. (The total number of NA structures for $d_{\mathrm{v}} = 3, 4, 5, 6$, $g = 6, 8$, and $a, b < 10$ is $51 + 178 + 23 = 252$.) Our goal is to investigate the LSS properties of these structures. In particular, we are interested in stripping down each structure to its non-cycle LSS-prime elementary subset. We start by NA structures of variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 6$. The first column of Table 3.1 shows that there are 51 such structures. The normal graphs for LSS-prime elementary trapping subsets of these 51 structures are shown in Figs. 3.3 and 3.4. Fig. 3.3 contains the subsets of size 6, 7 and 8. Size-9 subsets are presented in Fig. 3.4. We use the notation n_k^a for a non-cycle LSS-prime structure

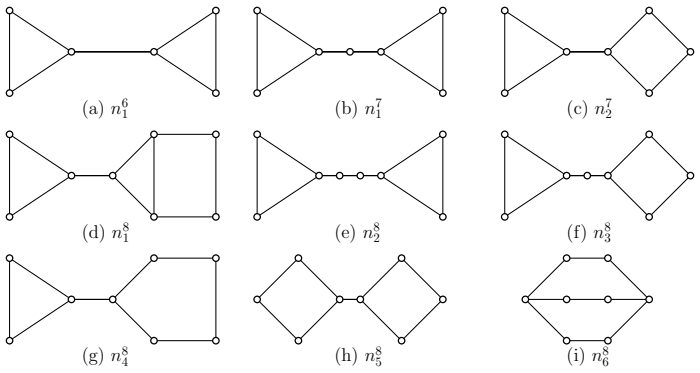


Figure 3.3: Normal graphs of all the non-isomorphic LSS-prime LETS structures of size 6, 7 and 8 within variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 6$. (a) n_1^6, (b) n_1^7, (c) n_2^7, (d) n_1^8, (e) n_2^8, (f) n_3^8, (g) n_4^8, (h) n_5^8, (i) n_6^8.

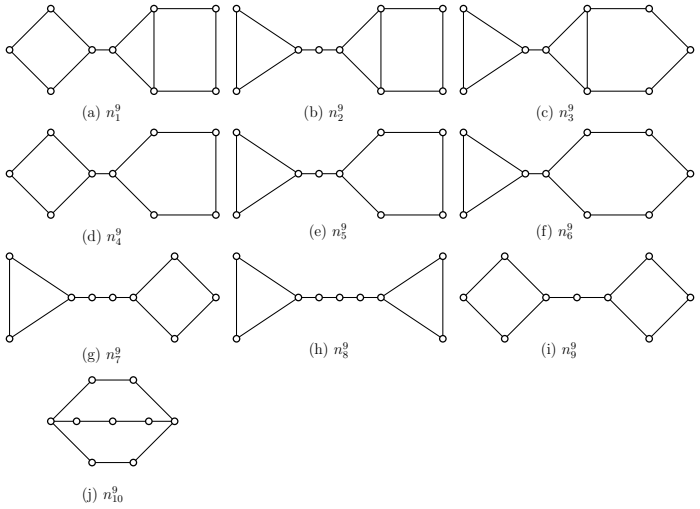


Figure 3.4: Normal graphs of all the non-isomorphic LSS-prime LETS structures of size 9 within variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 6$. (a) n_1^9, (b) n_2^9, (c) n_3^9, (d) n_4^9, (e) n_5^9, (f) n_6^9, (g) n_7^9, (h) n_8^9, (i) n_9^9, (j) n_{10}^9.

$\mathcal{S}$, where a is the number of variable nodes in $\mathcal{S}$, and k is a non-negative integer index that distinguishes among different non-isomorphic structures with the same a value. For example, the two LSS-prime structures with 7 variable nodes in Fig. 3.3 are denoted by n_1^7 and n_2^7.

It can be seen that all the 19 structures in Figs. 3.3 and 3.4 are composed of two cycles, each with maximum length 14. Starting from all the LETSs with these 19 structures in a Tanner graph, the LSS-based search algorithm will find all the LETSs

Table 3.2: LSS properties of NA structures for variable-regular graphs with $d_v = 3$ and $g = 6$

	$a = 6$	$a = 7$	$a = 8$	$a = 9$
b = 1	-	-	-	$n_1^7(1)$
b = 2	-	-	$n_1^6(1), n_1^7(1)$	-
b = 3	-	$n_1^6(1)$	-	$n_1^7(4), n_2^7(2), n_1^8(2), n_2^8(1), n_3^8(1)$
b = 4	$n_1^6(1)$	-	$n_1^7(2), n_2^7(2), n_1^8(1)$	-
b = 5	-	$n_1^7(1), n_2^7(1)$	-	$n_2^8(5), n_3^8(4), n_4^8(2), n_5^8(2)$ $n_6^8(1), n_1^9(1), n_2^9(1), n_3^9(1)$
b = 6	-	-	$n_2^8(1), n_3^8(1), n_4^8(1)$ $n_5^8(1), n_6^8(1)$	-
b = 7	-	-	-	$n_4^9(1), n_5^9(1), n_6^9(1)$ $n_7^9(1), n_8^9(1), n_9^9(1), n_{10}^9(1)$

with the 51 NA structures for variable-regular graphs with $d_v = 3$ and $g = 6$ listed in Table 3.1.

For variable-regular graphs with $d_v = 3$ and $g = 8$, there are five non-cycle LSS-prime structures, whose normal graphs are also included in Figs. 3.3 and 3.4: $\{n_5^8, n_6^8, n_4^9, n_9^9, n_{10}^9\}$. In Tables 8.2 and 3.3, we have listed the LSS-prime elementary subsets of NA structures for variable-regular graphs with $d_v = 3$, and $g = 6$ and $g = 8$, respectively. For example, the entry corresponding to $a = 8$ and $b = 4$ in Table 8.2 indicates that there are a total of 5 NA structures in the class of $(8, 4)$ LETSs, from which, two are LSSs of n_1^7, two are LSSs of n_2^7 and one is an LSS of n_1^8. (In fact, the last structure is n_1^8, itself.)

Table 3.3: LSS properties of NA structures for variable-regular graphs with $d_v = 3$ and $g = 8$

	$a = 8$	$a = 9$
b = 5	-	$n_5^8(1)$
b = 6	$n_5^8(1), n_6^8(1)$	-
b = 7	-	$n_4^9(1), n_9^9(1), n_{10}^9(1)$

Examining the 178 NA structures of variable-regular graphs with $d_v = 4$ and $g = 6$ (second column of Table 3.1), we find out that they are all LSSs of only 7 non-cycle LSS-prime structures. Four of these structures have normal graphs already included

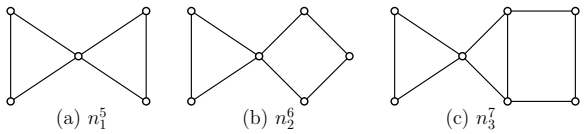


Figure 3.5: Normal graphs of 3 non-isomorphic LSS-prime LETS structures within variable-regular graphs with $d_v = 4$ and $g = 6$. (a) n_1^5, (b) n_2^6, (c) n_3^7.

Table 3.4: LSS properties of NA structures for variable-regular graphs with $d_v = 4$ and $g = 6$

		$a = 5$	$a = 6$	$a = 7$	$a = 8$	$a = 9$
b=4	LETS	-	-	-	$n_1^6(1)$	$n_1^5(1), n_1^6(4)$
	EAS	-	-	-	-	$n_1^5(1), n_1^6(1)$
b=6	LETS	-	-	$n_1^5(1)$	$n_1^5(3), n_1^6(4)$	$n_1^5(7), n_1^6(24), n_1^7(1), n_2^7(1)$
	EAS	-	-	-	$n_1^6(1)$	$n_1^6(3)$
b=8	LETS	$n_1^5(1)$	$n_1^5(1)$	$n_1^5(3), n_1^6(2)$	$n_1^5(6), n_1^6(13)$ $n_2^6(1)$	$n_1^5(15), n_1^6(67), n_2^6(5)$ $n_1^7(11), n_2^7(4), n_3^7(1), n_1^8(1)$
	EAS	-	-	-	-	$n_1^7(2)$

in the structures of Figs. 3.3(a),(b),(c),(d): $\{n_1^6, n_1^7, n_2^7, n_1^8\}$. (One should however, note that the fact that the variable degree in this case is 4 rather than 3 means that the same normal graph represents a different LETS structure compared to the case of $d_v = 3$.) The normal graphs of the other three structures are presented in Fig. 3.5.

In Table 3.4, we have listed the LSS-prime elementary subsets of NA structures of both LETSs and EASs for variable-regular graphs with $d_v = 4$ and $g = 6$.

For variable-regular graphs with $d_v = 5$ and $g = 6$, the 23 NA structures listed in Table 3.1 appear to be LSSs of only two LSS-prime structures. These structures have the same normal graph as n_1^6 and n_1^5 in Figs. 3.3(a) and 3.5(a), respectively. The LSS structure of the 23 NA structures in relation to these two LSS-prime structures is presented in Table 5.13. There is only one EAS structure in Table 5.13 for the class of $(9, 9)$ trapping sets. The result of Table 5.13 implies that starting from n_1^5, four successive applications of the LSS-based search algorithm will find this NA structure.

Table 3.5: LSS properties of NA structures for variable-regular graphs with $d_v = 5$ and $g = 6$

		$a = 8$	$a = 9$
b = 7	ETS:	-	$n_1^5(2), n_1^6(1)$
	EAS:	-	-
b = 8	ETS:	$n_1^5(1)$	-
	EAS:	-	-
b = 9	ETS:	-	$n_1^5(15), n_1^6(4)$
	EAS:	-	$n_1^5(1)$

Chapter 4

Dot-based Characterization/Search of LETSs of Variable-Regular LDPC Codes

4.1 Introduction

A careful examination of the LSS property in the space of normal graphs reveals that this property corresponds to a simple graph-based expansion technique, referred to, here, as *depth-one tree (dot)* expansion. In this chapter, dot-based characterization is studied.

4.2 Dot-based Characterization

The notation dot_m is used for a *dot* expansion with m edges, as shown in Fig. 4.1. In this figure, symbol $\circ$ is used to represent the m common nodes between the parent structure $\mathcal{S}$ and the dot_m structure. Symbol $\bullet$, on the other hand, is used to represent the root node of dot_m structure. For the dot_m expansion to result in a valid normal graph for a LETS structure of a variable-regular Tanner graph with variable degree $d_{\rm v}$, the m edges of the tree will have to be connected to nodes of $\mathcal{S}$ with degree less than $d_{\rm v}$. In addition, the degree m of the root node must be at least 2 and at most $d_{\rm v}$.

Proposition 1. *Suppose that $\mathcal{S}$ is an (a, b) LETS structure of variable-regular Tanner graphs with variable degree $d_{\rm v}$, where $b \geq 2$. Expansion of $\mathcal{S}$ using dot_m, $2 \leq m \leq min\{d_{\rm v}, b\}$, will result in LETS structure(s) in the $(a + 1, b + d_{\rm v} - 2m)$ class.*

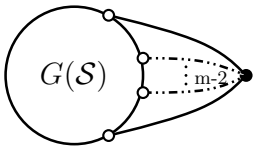


Figure 4.1: Expanding a LETS structure $\mathcal{S}$ by applying depth-one tree expansion with m edges (dot_m).

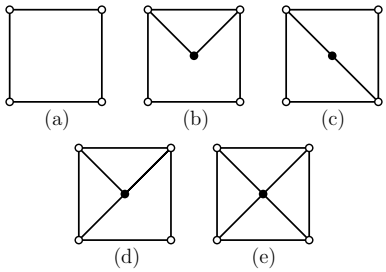


Figure 4.2: LETS structures with $\Delta(S) \leq d_{\mathrm{v}} = 4$: (a) The LETS structure $\mathcal{S}$ is in the $(4, 8)$ class, (b)-(c) LETS structures in the $(5, 8)$ class generated from $\mathcal{S}$ by dot_2 expansion, (d) LETS structure in the $(5, 6)$ class generated from $\mathcal{S}$ by dot_3 expansion, and (e) LETS structure in the $(5, 4)$ class generated from $\mathcal{S}$ by dot_4 expansion.

Proof. In addition to being limited by the upper bound of d_{v}, the maximum value of m is also upper bounded by b. Therefore, $m \leq min\{d_{\mathrm{v}}, b\}$. In the dot_m expansion, only one node is added to $\mathcal{S}$, and thus the size of the new structure is $a' = a + 1$. Based on Lemma 2, the number of edges in $\mathcal{S}$ is $|E_{\mathcal{S}}| = (ad_{\mathrm{v}} - b)/2$. By the nature of dot_m expansion, the number of edges in the expanded structure $\mathcal{S}'$ is $|E_{\mathcal{S}'}| = |E_{\mathcal{S}}| + m$. Therefore, based on Lemma 2, the number of unsatisfied check nodes b' in the new structure is $b' = (a + 1)d_{\mathrm{v}} - 2|E_{\mathcal{S}'}| = ad_{\mathrm{v}} - 2|E_{\mathcal{S}}| + d_{\mathrm{v}} - 2m = b + d_{\mathrm{v}} - 2m$. ■

Example 3. *Consider the LETS structure $\mathcal{S}$ of Fig. 4.2(a) in a variable-regular Tanner graph with $d_{\mathrm{v}} = 4$ and $g = 6$. This structure is in the $(4, 8)$ class. The application of dot_2 expansion to $\mathcal{S}$ generates two non-isomorphic structures shown in Figs. 4.2(b) and (c) in the $(5, 8)$ class. For each of the two expansions dot_3 and dot_4 applied to $\mathcal{S}$, only one new structure is generated. This structure belongs to the $(5, 6)$ and the $(5, 4)$ class, respectively, and is shown in Figs. 4.2(d) and (e), respectively.*

Proposition 2. *Among the LETS structures with the same size a, the simple cycle of size a in the $(a, a(d_{\mathrm{v}} - 2))$ class has the largest b.*

Proof. Since $b = ad_v - 2|E|$, the LETS structure with the largest b is the one with the minimum number of edges $|E|$. Also, in leafless structures, each node has at least two incident edges. Therefore, the simple cycle of size a, in which each node has exactly two incident edges, has the minimum number of edges, a, among all the LETS structures with the same size. The simple cycle of size a is, in fact, the only LETS structure in the $(a, ad_v - 2a = a(d_v - 2))$ class. ■

Remark 1. *In a Tanner graph with girth g, the smallest LETS structure is the simple cycle of size $g/2$, denoted by $s_{g/2}$. The $s_{g/2}$ structure is the only LETS structure of size $g/2$.*

Given all the non-isomorphic LETS structures of size k, the *dot* expansion technique is able to generate most of the non-isomorphic LETS structures of size $k+1$. The LETS structures which cannot be obtained by the *dot* expansion from any of their subsets are called *dot-prime structures* (LSS-prime structures in Chapter 3). For brevity, we refer to these structures as being *prime*, in the rest of the work. Prime structures can be categorized into: (i) *cycle prime structures*, and (ii) *non-cycle prime structures*. In the first category, we have all the simple cycles, and cycles with chord that are out of the reach of the *dot* expansion. In the second category, we have structures that are slightly more complex than cycles. Non-cycle prime structures are discussed in details in Chapter 3.

Dot expansion is a simple recursive technique to generate larger LETS structures from smaller ones. Based on Remark 1, one needs to start from the simple cycle of length $g/2$ and apply the *dot* expansion recursively, to obtain non-isomorphic LETS structures in different (a, b) classes with $a > g/2$. This technique, however, has the limitation of leaving out all the prime structures. Therefore, for a full characterization of LETS structures based on *dot* expansion, these prime structures will have to be identified and added as new initial building blocks for the *dot* expansion.

Remark 2. *Expanding LETS structures with depth-one tree is the same as the LSS-based algorithm proposed in [46]. For the characterization of LETS structures, however, rather than examining each structure in a class to find out its LSS properties, as performed in [46], in this work, the non-isomorphic LETS structures are generated by expanding the prime structures using the dot expansion technique.*

Case Study-*Characterization of non-isomorphic LETS structures of (a, b) classes for variable-regular graphs with $d_v = 3$, $g = 6$ and $a \leq 9, b \leq 9$:*

Table 4.1: dot-based characterization of non-isomorphic LETS structures of (a, b) classes for variable-regular graphs with $d_v = 3$, $g = 6$ and $a \leq 9, b \leq 9$

	$a=3$	$a=4$	$a=5$	$a=6$	$a=7$	$a=8$	$a=9$
b = 0	-	$s_3(1)$	-	$s_4(2)$	-	$s_5(3), s_6(1), c_6(1)$	-
b = 1	-	-	$s_3(1)$	-	$s_4(3), s_5(1)$	-	$s_5(9), s_6(5)$ $c_6(2), n_6(1)$ $c_7(1), n_7(1)$
b = 2	-	$s_3(1)$	-	$s_4(3), s_5(1)$	-	$s_5(9), s_6(5), c_6(2)$ $n_6(1), c_7(1), n_7(1)$	-
b = 3	$\boldsymbol{s_3(1)}$	-	$s_4(2)$	-	$s_5(6), s_6(1)$ $c_6(2), n_6(1)$	-	$s_6(31), s_7(9)$ $c_7(9), n_7(7)$ $c_8(3), n_8(4)$
b = 4	-	$\boldsymbol{s_4(1)}$	-	$s_5(2), \boldsymbol{c_6(1)}$ $\boldsymbol{n_6(1)}$	-	$s_6(12), s_7(2), c_7(4)$ $n_7(4), \boldsymbol{c_8(2)}, \boldsymbol{n_8(1)}$	-
b = 5	-	-	$\boldsymbol{s_5(1)}$	-	$s_6(3), \boldsymbol{c_7(1)}$ $\boldsymbol{n_7(2)}$	-	$s_7(19), s_8(2)$ $c_8(11), n_8(15)$ $\boldsymbol{c_9(2)}, \boldsymbol{n_9(3)}$
b = 6	-	-	-	$\boldsymbol{s_6(1)}$	-	$s_7(3), \boldsymbol{c_8(2)}, \boldsymbol{n_8(5)}$	-
b = 7	-	-	-	-	$\boldsymbol{s_7(1)}$	-	$s_8(4), \boldsymbol{c_9(2)}$ $\boldsymbol{n_9(7)}$
b = 8	-	-	-	-	-	$\boldsymbol{s_8(1)}$	-
b = 9	-	-	-	-	-	-	$\boldsymbol{s_9(1)}$

This characterization is summarized in Table 4.1, where the prime structures are boldfaced. The notations s_k, c_k, and n_k are used to denote the simple cycle, a prime cycle with chord, and a non-cycle prime structure of size k, respectively. As an example of how the entries in Table 4.1 should be interpreted, we consider the $(7, 5)$ class. The entries for this class are: $s_6(3), c_7(1)$, and $n_7(2)$. This means that there are a total of 6 non-isomorphic LETS structures in this class, three of these structures are generated, using *dot* expansions, starting from s_6, one of them is a prime cycle with chord of size 7, and two of them are non-cycle prime structures of size 7. Having the symbol "-" for an (a, b) class means that it is impossible to have any LETS structure in that class. In Table 4.1, the smallest possible LETS structure is the cycle of size 3, s_3. The two non-isomorphic LETS structures of size 4 generated by the application of *dot* expansions to s_3 are shown in Fig. 4.3 (f_4 is the new node added to s_3).

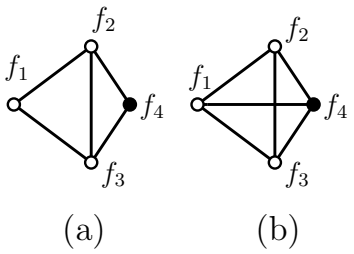


Figure 4.3: LETS structures in (a) $(4, 2)$ and (b) $(4, 0)$ classes in variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 6$.

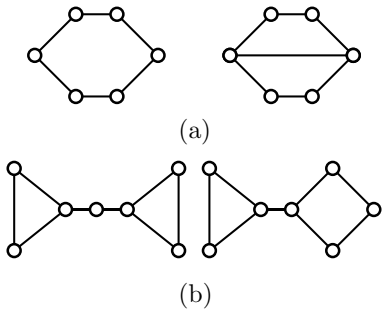


Figure 4.4: Prime LETS structures in variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 6$ (a) a simple cycle in the $(6, 6)$ class and a cycle with chord in the $(6, 4)$ class (b) non-cycle prime structures in the $(7, 5)$ class.

These structures, which belong to $(4, 2)$ and $(4, 0)$ classes, respectively, are the only non-isomorphic LETS structures of size 4 that are generated by the *dot* expansion. There, however, remains another LETS structure of size 4 in the $(4, 4)$ class that is not generated by the *dot* expansion. This $(4, 4)$ LETS structure, which is in fact s_4, is thus chosen as a prime structure.

Fig. 4.4(a) shows two prime structures in the $(6, 6)$ and $(6, 4)$ classes, respectively, and Fig. 4.4(b) depicts two prime structures in the $(7, 5)$ class.

It is worth noting that unlike [46] and Chapter 3, the simple cycles and prime cycles with chord are distinguished in this thesis.

4.3 Dot-based Search

Dot-based characterization of LETS structures corresponds to a search algorithm to find all the instances of LETS structures in the Tanner graph of LDPC codes for

given values of d_v, g, a_{max} and b_{max}. First, based on the characterization table, the set of prime structures in the interest range of $a \leq a_{max}$ and $b \leq b_{max}$ are identified. For example, for variable-regular graphs with $d_v = 3$, $g = 6$ and $b_{max} = a_{max} = 9$, all the prime structures of Table 4.1 are needed. As another example, if, for a variable-regular graph with $d_v = 3$ and $g = 6$, one is interested in $b_{max} = 3$ and $a_{max} = 8$, only 8 prime structures in the $(3,3)$, $(4,4)$, $(5,5)$, $(6,4)$, $(6,6)$ and $(7,5)$ classes are needed. Suppose that T prime structures are needed for $a \leq a_{max}$ and $b \leq b_{max}$. The instances of each prime structure are then enumerated in the Tanner graph of the code, and are used as the input to the dot-based search algorithm to find all the instances of LETS structures in the interest range of $a \leq a_{max}$ and $b \leq b_{max}$. This dot-based search algorithm is essentially the same as the LSS-based search algorithm of [46].

4.4 Shortcomings of Dot-based Search Algorithm

To motivate the new characterization/search technique, we start by explaining the main practical issues in implementing a dot-based search algorithm. In summary, in a given Tanner graph, to exhaustively search for (a, b) LETS instances within the range of interest $a \leq a_{max}$ and $b \leq b_{max}$, one needs to initially enumerate instances of prime structures that are needed as parents of the LETS structures under consideration. Some of these prime structures have relatively large a and b values. For practical codes, the multiplicity of instances of such prime structures can be easily in the range of tens of millions. This imposes a huge computational burden and memory requirement on the dot-based search algorithm. The dot-based search algorithm in this case is particularly inefficient, when such prime structures, themselves, are not of direct interest (as they have relatively large a and b values), and when the multiplicity of the instances of the LETS structures, that are children of such prime structures and of interest, is relatively low, or in some cases even zero.

We consider the case of variable-regular graphs with $d_v = 3$ and $g = 6$. For such graphs, the dot-based characterization for the range of $a \leq 9$ and $b \leq 9$, was presented in Table 4.1. In Table 4.2, we have extended the results of Table 4.1 to cover LETS structures with $a = 10, 11, 12$, and $b \leq 5$. All the (non-cycle and cycle) prime structures of size less than 10 (those identified in Table 4.1 by s_k, c_k, n_k, with $k \leq 9$) are used to generate the LETS structures in Table 4.2. For the LETS structures

Table 4.2: dot-based characterization of non-isomorphic LETS structures of (a, b) classes for variable-regular graphs with $d_v = 3$, $g = 6$, $10 \leq a \leq 12$ and $b \leq 5$

	$a = 10$	$a = 11$	$a = 12$
b = 0	$s_6(12), s_7(3), c_7(2)$ $n_7(1), n_8(1)$	-	$c_7(43), s_8(15), c_8(12)$ $n_8(9), s_9(1), c_9(1), n_9(4)$
b = 1	-	$s_6(50), s_7(26), c_7(10), n_7(7)$ $s_8(4), c_8(6), n_8(9), n_9(2)$	-
b = 2	$s_6(50), s_7(24), c_7(10), n_7(7)$ $s_8(3), c_8(8), n_8(10), n_9(1)$	-	$s_7(350), s_8(156), c_8(93)$ $n_8(73), s_9(22), c_9(44)$ $n_9(77), \mathbf{NA(20)}$
b = 3	-	$s_7(211), s_8(65), c_8(72), n_8(65)$ $s_9(4), c_9(21), n_9(41), \mathbf{NA(3)}$	-
b = 4	$s_7(75), s_8(20), c_8(37), n_8(37)$ $c_9(10), n_9(18), \mathbf{NA(1)}$	-	$s_8(711), s_9(202), c_9(322)$ $n_9(346), \mathbf{NA(311)}$
b = 5	-	$s_8(172), s_9(40), c_9(104)$ $n_9(139), \mathbf{NA(81)}$	-

which are not generated using the *dot* expansion technique starting from any of these prime structures, the notation "NA" is used in Table 4.2. Starting from the prime structures of size 10 and 11, one can also generate (characterize) NA LETS structures of Table 4.2.

Example 4. *The* $(11, 1)$ *class is a potentially dominant LETS class of variable-regular graphs with* $d_v = 3$ *and* $g = 6$. *Table 4.2 shows that among 114 non-isomorphic LETS structures in this class, 50, 26, 10, 7, 4, 6, 9 and 2 structures are generated starting from prime structures* s_6, s_7, c_7, n_7, s_8, c_8, n_8 *and* n_9, *respectively. This means prime structures with size up to* 9 *need to be enumerated in the Tanner graph to find the instances of all the LETS structures of this class (if any exists). Fig. 4.5 represents* $\mathcal{S} = (\{f_1, f_2, \ldots, f_{11}\}, E)$, *a LETS structure in the* $(11, 1)$ *class which is generated starting from the simple cycle prime structure,* s_8, *in the* $(8, 8)$ *class* $(\mathcal{S}' = (\{f_1, f_2, \ldots, f_8\}, E'))$. *To find all the instances of* $\mathcal{S}$ *in the Tanner graph using a dot-based search algorithm, all the instances of* s_8 *in the graph need to be enumerated.*

Example 5. *In the* $(12, 2)$ *class of Table 4.2, in addition to LETS structures that are generated by the dot-based expansion starting from* c_7, s_8, c_8, n_8, s_9, c_9 *and* n_9,

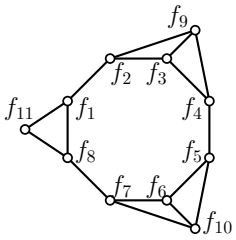


Figure 4.5: A LETS structure in the $(11, 1)$ class of a variable-regular graph with $d_v = 3$ and $g = 6$.

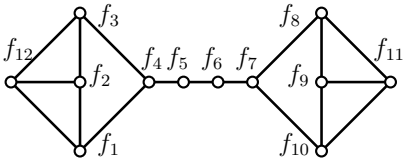


Figure 4.6: A LETS structure in the $(12, 2)$ class of a variable-regular graph with $d_v = 3$ and $g = 6$.

there are 20 non-isomorphic LETS structures that are generated from prime structures of size 10 and 11 (denoted as NA in the table). Fig. 4.6 shows the structure $\mathcal{S} = (\{f_1, f_2, \ldots, f_{12}\}, E)$*, as one of these NA structures. This structure is generated starting from a non-cycle prime structure of size 10 in the* $(10, 8)$ *class* $(\mathcal{S}' = (\{f_1, f_2, \ldots, f_{10}\}, E'))$*. Although a certain Tanner graph with* $d_v = 3$ *and* $g = 6$ *may not contain any instance of* $\mathcal{S}$*, all the instances of the structure* $\mathcal{S}'$ *in the Tanner graph need to be enumerated for the dot-based search algorithm to guarantee the exhaustive coverage of the* $(12, 2)$ *class.*

Example 6. *Consider a* $(3, 6)$*-regular LDPC code with* $g = 6$*, and* $n = 20000$ *[57].*

Table 4.3: Multiplicities of prime structures of size 8 and 9 in the code of Example 6

Prime structure	Multiplicity	Prime structure	Multiplicity
c_8 in $(8, 4)$ class	0	c_9 in $(9, 5)$ class	15
c_8 in $(8, 6)$ class	6430	c_9 in $(9, 7)$ class	83392
s_8 in $(8, 8)$ class	6219941	s_9 in $(9, 9)$ class	55181079
n_8 in $(8, 4)$ class	0	n_9 in $(9, 5)$ class	1
n_8 in $(8, 6)$ class	6754	n_9 in $(9, 7)$ class	101777

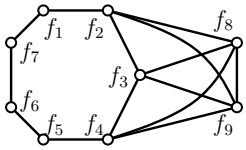


Figure 4.7: A LETS structure in the $(9, 8)$ class of a variable-regular graph with $d_{\mathrm{v}} = 4$ and $g = 6$.

Suppose that one is interested in LETSs of this code in the range of $a \leq 12$ *and* $b \leq 5$. *The multiplicities of all the instances of prime structures of size 8 and 9 in the Tanner graph of this code in the* $(8, 4)$, $(8, 6)$, $(8, 8)$, $(9, 5)$, $(9, 7)$ *and* $(9, 9)$ *classes are listed in Table 4.3. In the range* $a \leq 12$ *and* $b \leq 5$, *the* $(8, 4)$, $(9, 3)$, $(9, 5)$, $(10, 0)$, $(10, 2)$, $(10, 4)$, $(11, 1)$, $(11, 3)$, $(11, 5)$, $(12, 0)$, $(12, 2)$, $(12, 4)$ *classes are the classes that can have LETS structures starting from these prime structures. To exhaustively search the Tanner graph for LETSs with* $a \leq 12$ *and* $b \leq 5$, *the dot-based search algorithm will have to enumerate all the instances of the prime structures listed in Table 4.3 (a total of more than 60 million). In all the above classes, however, this code has only 15, 1 and 8 instances of LETS structures in* $(9, 5)$, $(10, 4)$ *and* $(11, 5)$ *classes, respectively, that are generated by the dot-based expansion, starting from the instances of these (more than 60 million) prime structures.*

Example 7. *Consider a* $(4, 8)$*-regular LDPC code with* $g = 6$, *and* $n = 4000$ *[57].Suppose that one is interested in finding all the LETSs of this code in the range* $a \leq 9$ *and* $b \leq 8$, *using the dot-based search algorithm. To do this, based on Table V of [46], one must enumerate all the instances of simple cycle prime structure,* s_7. *The LETS structure in Fig. 4.7 is an example of a structure in the* $(9, 8)$ *class that has* s_7 *as its prime sub-structure. It appears that while there are* $123,111,331$ *instances of* s_7 *in the Tanner graph of this code, there is not a single instance of a LETS structure in all the classes of interest which has* s_7 *as its prime sub-structure, in the Tanner graph.*

Chapter 5

Dpl-based Characterization/Search of LETSs of Variable-Regular LDPC Codes

5.1 Introduction

In this chapter, first we provide a general framework for exhaustive search of LETSs. Then, to overcome the shortcomings of the dot-based search algorithm, explained in Section 4.4, we propose two new graph expansion techniques, *path* and *lollipop* expansions, that are applied to smaller LETS structures to generate larger ones. Similar to *dot* expansion, the new expansions are also applied in the space of normal graphs. In Section 5.4, we then use *path* and *lollipop* expansions along with the *dot* expansion, to formulate a new characterization of LETS structures. In the context of characterizing a LETS structure as a sequence of embedded LETS structures that starts from a simple cycle and expands, step by step, to reach the structure of interest, we introduce the concept of *minimal* characterizations, as those in which none of the expansions in the sequence can be divided into smaller ones. The proposed characterization in Section 5.4 is minimal, and describes each and every LETS structure as an expansion of a simple cycle, using a combination of *dot*, *path* and *lollipop* expansion techniques, thus the name *dpl-based characterization*, or *dpl characterization*, in brief. In Section 5.4, we also prove that any minimal characterization is based only on the expansions *dot*, *path* and *lollipop*, and that the proposed characterization has some optimal properties as related to the corresponding search algorithm. The characterization results for some values of $d_{\mathrm{v}}, g, a_{max}$, and b_{max} are presented as characterization tables in Section 5.5. For a given range $a \leq a_{max}$ and $b \leq b_{max}$, the maximum length of simple cycles participating in the *dpl* characterization is often less than that for the *dot*

characterization. In Section 5.6, we develop the search algorithm corresponding to the *dpl* characterization, and in Section 5.7, we discuss the complexity of the search. Finally numerical results are presented in Section 5.8.

5.2 Exhaustive Search Framework for LETSs based on the Hierarchical Relationship of LETS Graphical Structures

Suppose that one is interested in finding all the instances of (a, b) LETS structures of a given Tanner graph within the range $a \leq a_{max}$ and $b \leq b_{max}$. A crude and highly complex approach would be to consider any combination of a variable nodes in the Tanner graph and examine it to see whether it is an (a, b) LETS with $b \leq b_{max}$. This process will have to be repeated for all the values of $a \leq a_{max}$. A more efficient approach would be to first find all the non-isomorphic structures of LETSs within the interest range of a and b values, and then search for each structure, one at a time, within the Tanner graph. Although more efficient than the first approach, this method is also too complex to be useful in practical settings. To further simplify the exhaustive search, one can carefully examine the graphical structure of all the non-isomorphic LETSs of interest and look for hierarchical (parent-child) relationships that can be used to simplify the overall search. Suppose that L_1 and L_2 are two LETS structures of interest, and that L_1 is a sub-structure (parent) of L_2, i.e., $L_1 \subset L_2$. To find all the instances of L_1 and L_2, one can first perform an exhaustive search to find all the instances of L_1. Then, instead of performing a new exhaustive search to find all the instances of L_2, one can just check whether any instance of L_1 can be expanded to an instance of L_2 by grafting the expansion $L_2 \backslash L_1$ into L_1.

Based on the above principle, a general framework for the exhaustive search of (a, b) LETSs with $a \leq a_{max}$ and $b \leq b_{max}$ is as follows: Suppose that there are N non-isomorphic LETS structures $L_1, \ldots, L_N$, within the range of interest. (These structures can be efficiently found using the normal graph representation and *nauty*.) Also, suppose that these structures are organized in a collection of rooted trees (forest) where each structure is represented by a node in a tree and the parent-child relationships between substructures are represented by edges, i.e., if $L_i \subset L_j$, and if we plan to use the exhaustive search of L_i as a starting point for the exhaustive search

of L_j, as described above, then there is an edge between the corresponding nodes. In this case, the node for L_i is called the parent (immediate ancestor) of the node for L_j, and is closer to the root of the tree than the node for L_j is. Given such a forest, one starts by finding all the instances of the root structures. The instances of all the other structures within each tree are then found by searching for the expansions of the instances of their parent structure. This will be done recursively, where each structure at depth i of a tree is searched for, as an expansion of its parent structure at depth $i-1$ of the tree. In this framework, given that the search of the root structures is exhaustive, it is clear that the search for all the remaining structures will also be exhaustive. The search framework just presented can be further generalized by adding some auxiliary structures to the forest. These new structures, which may be LETS structures outside the range of interest or structures that are not even LETS, would be added to help in simplifying the overall search. For example, they may create links between two LETS structures of interest that have a hierarchical relationship, but through a complex expansion. The introduction of the auxiliary structure(s) can break the complex expansion into simpler ones.

The efficiency of an specific exhaustive search algorithm within the above framework depends highly on the design of the forest. In general, one would like to have a forest with smaller number of trees, where the roots are simple structures that can be efficiently enumerated, and in each tree, the expansion relationships between the structures are relatively simple.

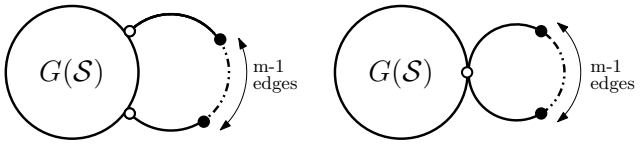
In [47], Karimi and Banihashemi proposed a recursive search, similar to the framework described above, to find LETS instances within a Tanner graph. They used short cycles as the roots of the trees for variable-regular Tanner graphs, and short cycles plus variable nodes with low degree, and cycles with low approximate cycle extrinsic message degree (ACE) as the roots of the trees for the case of irregular Tanner graphs. They also described the expansions relating two LETS structures in terms of the size difference between the two structures, and used those expansions to organize the recursive search. The search algorithm of [47], although efficient, did not provide any guarantee that the search results were exhaustive. In [46], Karimi and Banihashemi focused on variable-regular Tanner graphs and a simple hierarchical relationship between LETS structures, dubbed LSS (layered super set) property, and devised a recursive search algorithm based on trees, each rooted at short cycles of a certain length and being connected internally based on the LSS relationship between

Table 5.1: Pros and cons of the search algorithms in [47], [46], Chapter 3 and this chapter

Goals	Exhaustive Search	Simple roots	Simple expansions	Small # trees	Small # auxiliary nodes
[47]	✗	✓	✓	✓	✓
[46]	✗	✓	✓	✓	✗
Chapter 3	✓	✗	✓	✗	✗
Chapter 5	✓	✓	✓	✓	✓

their nodes. The forest used in the search algorithm of [46] has the advantage that the expansions are very simple, but suffers from the introduction of some auxiliary structures that may be very complex and abundant. It also lacks the guarantee for exhaustiveness. This latter shortcoming is resolved in Chapter 3, where new trees were added to the forest of [46]. The new trees were rooted at some auxiliary LETS structures but were still formed following the LSS-based expansion as the relationship between their nodes.

In this work, following the above general framework for exhaustive search, we design a forest whose trees are each rooted at a simple short cycle of a different length. This makes the number of rooted trees small. It also makes the enumeration of the instances of the root structure for each tree simple. The expansions within each tree are limited to three types, one being the LSS-based expansion of [46]. The expansions in each tree are selected to be minimal in the sense that they cannot be broken into smaller (simpler) expansions such that the intermediate structures still remain LETS. This satisfies the constraint of having simple expansions in the search process. Finally, auxiliary LETS structures are added systematically as intermediate or root nodes of the trees, as required, with the goal of minimizing the overall search complexity. Table 5.1 summarizes the pros and cons of the search algorithms in [47], [46], Chapter 3 and this chapter.



(a) (b)

Figure 5.1: Expansion of the LETS structure $\mathcal{S}$ with (a) an open *path* of length $m+1, pa_m^o$ (b) a closed *path* of length $m+1, pa_m^c$.

5.3 *Path* and *Lollipop* Expansion Techniques

Consider an (a, b) LETS structure $\mathcal{S}$ of a d_{v}-regular Tanner graph with $g \geq 6$. The *path* expansion of $\mathcal{S}$ is a new LETS structure $\mathcal{S}'$ of size $a+m$, that is constructed by appending a path of length $m+1$ to $\mathcal{S}$. The first and the last nodes of the path are common with $\mathcal{S}$, and can be identical, in that case, the path is closed. Figures 5.1(a) and (b), show the *path* expansion of $\mathcal{S}$ using open and closed paths of length $m+1$, respectively. In these figures, the symbol $\circ$ is used to represent the common node(s) between $\mathcal{S}$ and the path, and the symbol $\bullet$ is used to represent the other m nodes of the path. It is clear that for an *open-path* expansion, the degrees of the two nodes that are common with $\mathcal{S}$ must be strictly less that d_{v} (in $G(\mathcal{S})$), and for the *closed-path* expansion, the degree of the one common node must be strictly less than $d_{\text{v}} - 1$ in $G(\mathcal{S})$. We use the notations pa_m^o and pa_m^c for open and closed paths of length $m+1$, respectively. The notation pa_m is used to include both open and closed paths.

Proposition 3. *Suppose that $\mathcal{S}$ is a LETS structure in the (a, b) class ($b \geq 2$) for variable-regular Tanner graphs with variable degree d_{v}. Applying the path expansion pa_m, with $m \geq 2$, to $\mathcal{S}$ will result in LETS structure(s) in the $(a+m, b-2+m(d_{\text{v}}-2))$ class.*

Proof. The constraint $b \geq 2$ is to accommodate the 2 edges of the path that are connected to $\mathcal{S}$. We also note that pa_1^o is the same as dot_2, and that the expansion pa_1^c will result in a cycle of length 4. We thus focus on $m \geq 2$ in the statement of the proposition.

The expanded structure $\mathcal{S}'$ has size $a' = a + m$. Based on Lemma 2, the number of edges in $\mathcal{S}$ is $|E_{\mathcal{S}}| = (ad_{\text{v}} - b)/2$. By the nature of expansion, the number of edges

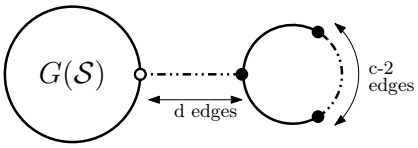


Figure 5.2: Expansion of the LETS structure $\mathcal{S}$ with a lollipop walk of length $m+1 = d + c$, lo^c_m.

in $\mathcal{S}'$ is $|E_{\mathcal{S}'}| = |E_{\mathcal{S}}| + m + 1$. By combining these and using Lemma 2, we have $b' = a'd_{\rm v} - 2|E_{\mathcal{S}'}| = (a+m)d_{\rm v} - (ad_{\rm v} - b) - 2m - 2 = b - 2 + m(d_{\rm v} - 2)$. ■

The pseudo-code of the *path* expansion algorithm is given in Routine 1.

Routine 1 (PathExpansion) Expansion of a LETS structure $\mathcal{S}$ to all possible LETS structures $\mathcal{L}^p$ of size $|\mathcal{S}| + m$, using pa_m. $\mathcal{L}^p = \text{PathExpansion}(\mathcal{S}, m)$

1: **Initialization:** Identify the subset of nodes in $\mathcal{S}$ with degree strictly less that $d_{\rm v}$, and store them in F'. Identify the subset of nodes in $\mathcal{S}$ with degree strictly less than $d_{\rm v} - 1$, and store them in F''. $\mathcal{L}^p \leftarrow \emptyset$.
2: Construct $\mathcal{N}_2(G)$ to include all the possible subsets of F' of size 2.
3: **for** each set in $\mathcal{N}_2(G)$ **do**
4: Generate a new LETS structure $\mathcal{S}'$ by applying pa^o_m expansion to $\mathcal{S}$ such that the 2 nodes in F' are the first and the last nodes of the new path of length $m+1$.
5: $\mathcal{L}^p \leftarrow \mathcal{L}^p \cup \mathcal{S}'$.
6: **end for**
7: **for** each node in F'' **do**
8: Generate a new LETS structure $\mathcal{S}''$ by applying pa^c_m expansion to $\mathcal{S}$ such that the node in F'' is the common node between $\mathcal{S}$ and the new path of length $m+1$.
9: $\mathcal{L}^p \leftarrow \mathcal{L}^p \cup \mathcal{S}''$.
10: **end for**
11: **Output:** $\mathcal{L}^p$.

In Fig. 5.2, the expansion of a LETS structure $\mathcal{S}$ using a lollipop walk of length $m+1$ is shown. The notation lo^c_m is used for a lollipop walk of length $m+1$ (m is number of the nodes added to $\mathcal{S}$), which consists of a cycle of length c ($c \geq 3$) and a path of length d ($d \geq 1$). Clearly, the common node between $\mathcal{S}$ and the *lollipop* expansion must have a degree strictly less than $d_{\rm v}$ in $G(\mathcal{S})$.

Proposition 4. *Suppose that $\mathcal{S}$ is a LETS structure in the (a, b) class $(b \geq 1)$ for variable-regular Tanner graphs with variable degree d_{v}. Applying the lollipop expansion lo_m^c, with $m \geq 3$ and $3 \leq c \leq m$, to $\mathcal{S}$ will result in LETS structure(s) in the $(a + m, b - 2 + m(d_{\mathrm{v}} - 2))$ class.*

Proof. Similar to that of Proposition 3. ■

The pseudo-code for the *lollipop* expansion is given in Routine 2.

Routine 2 (LollipopExpansion) Expansion of a LETS structure $\mathcal{S}$ to all possible LETS structures $\mathcal{L}_c^l$ of size $|\mathcal{S}| + m$, using lo_m^c. $\mathcal{L}_c^l = \text{LollipopExpansion}(\mathcal{S}, m, c)$

1: **Initialization:** Identify the subset of nodes in $\mathcal{S}$ with degree strictly less than d_{v} and store them in F'. $\mathcal{L}_c^l \leftarrow \emptyset$.
2: **for** each node f in F' **do**
3: Generate a new LETS structure, $\mathcal{S}'$, by expanding $\mathcal{S}$ using lo_m^c expansion such that f is the common node between the lollipop and $\mathcal{S}$.
4: $\mathcal{L}_c^l \leftarrow \mathcal{L}_c^l \cup \mathcal{S}'$.
5: **end for**
6: **Output:** $\mathcal{L}_c^l$.

In the following, we demonstrate, through a sequence of intermediate results, that any LETS structure of variable-regular Tanner graphs can be generated from simple cycles using a combination of the three expansion techniques: *dot*, *path* and *lollipop*.

Lemma 3. *Except the simple cycles, any LETS structure $\mathcal{S}' = (F', E')$ contains at least one LETS sub-structure $\mathcal{S} = (F, E)$ with $|F| < |F'|$.*

Proof. Since the structure $\mathcal{S}'$ is leafless, it must contain a cycle, and as a result, a simple cycle. The proof then follows from the fact that simple cycles are LETSs. ■

Lemma 4. *Suppose that a LETS structure $\mathcal{S}' = (F', E')$ has a LETS sub-structure $\mathcal{S} = (F, E)$, where $|F| = |F'| - 1$. Structure $\mathcal{S}'$ can then be generated from $\mathcal{S}$ by using the dot expansion.*

Proof. Node f in $F' \backslash F$ must be connected to at least two nodes in $\mathcal{S}$. Based on the definition of *dot* expansion, structure $\mathcal{S}'$ can then be generated from $\mathcal{S}$ by using dot_m expansion $(m \geq 2)$, where the root of the expansion tree is f. ■

Lemma 5. *Consider a LETS structure $\mathcal{S}' = (F', E')$ that is prime with respect to dot expansion, and is not a simple cycle. Then, for the largest LETS sub-structure $\mathcal{S} = (F, E)$ of $\mathcal{S}'$, we have*
(i) $|F| < |F'| - 1$,
(ii) subgraph induced by $F' \backslash F$ is connected,
(iii) no node in $F' \setminus F$ can have more than one edge connected to the nodes in $\mathcal{S}$.

Proof. Proof of (i) follows from Lemmas 3 and 4, and the definition of a prime structure. For (ii), assume that the subgraph is disconnected. Then removing the nodes in one of the disconnected parts from F' results in a new LETS sub-structure of $\mathcal{S}'$, which is larger than $\mathcal{S}$. This contradicts the assumption that $\mathcal{S}$ is the largest LETS sub-structure of $\mathcal{S}'$. For (iii), again by contradiction, if such a node exists, by removing all the other nodes in $F' \setminus F$, we obtain a LETS sub-structure of $\mathcal{S}'$ larger than $\mathcal{S}$. ■

Proposition 5. *Suppose that $\mathcal{S}' = (F', E')$ is a prime structure of dot expansion, but is not a simple cycle. Let $\mathcal{S} = (F, E)$ be the largest LETS sub-structure of $\mathcal{S}'$. Structure $\mathcal{S}'$ can then be generated by expanding $\mathcal{S}$ using a path or a lollipop walk.*

Proof. Based on Lemma 5, in the following, we consider two cases of $|F| = |F'| - 2$ and $|F| < |F'| - 2$.

Case (1)–$|F| = |F'| - 2$: Based on Part (ii) of Lemma 5, the two nodes in $F' \setminus F$ must be adjacent. Moreover, based on Part (iii) of the same lemma, each node in $F' \setminus F$ must have one connection to the nodes in $\mathcal{S}$. The only possible configurations satisfying both conditions are the expansions of $\mathcal{S}$ by the closed and open paths of length 3.

Case (2)–$|F| < |F'| - 2$: We first prove that in this case, the number of the edges connecting the nodes in $F' \setminus F$ to the nodes in $\mathcal{S}$ must be strictly less than 3. Suppose that the number of such edges is at least 3. This means that, based on Part (iii) of Lemma 5, there are at least three nodes in $F' \setminus F$ each connected with one edge to a node in $\mathcal{S}$. Select three such nodes, and call them v_1, v_2 and v_3. Based on Part (ii) of Lemma 5, there is a path between any pair of these nodes in the subgraph induced by $F' \setminus F$. Find the shortest paths between any pair of these nodes in the subgraph. Without loss of generality, assume that among the three shortest paths, the one between v_1 and v_2 has the smallest length. Clearly v_3 cannot be on this path. Add the nodes v_1, v_2 and all the nodes on the shortest path between v_1 and v_2 to $\mathcal{S}$.

This results in a LETS sub-structure of $\mathcal{S}'$ that is larger than $\mathcal{S}$ and strictly smaller than $\mathcal{S}'$, which is a contradiction. Therefore, the number of the edges connecting the nodes in $F' \setminus F$ to the nodes in $\mathcal{S}$ must be either two or one.

For the case where there are two edges connecting $F' \setminus F$ to $\mathcal{S}$, based on Part (iii) of Lemma 5, these edges must be incident to two distinct nodes in $F' \setminus F$. Call these nodes v_1 and v_2. There must be no cycles in the subgraph induced by $F' \setminus F$. Otherwise, one can find the shortest path between v_1 and v_2 in the subgraph and by removing all the nodes in $F' \setminus F$ except v_1, v_2 and all the nodes on the shortest path between them, obtain a LETS sub-structure of $\mathcal{S}'$ that is larger than $\mathcal{S}$ but strictly smaller than $\mathcal{S}'$. Therefore, for this case, the only possible configurations are the expansions of $\mathcal{S}$ by closed and open paths of length more than 3.

For the case where there is only one edge connecting $F' \setminus F$ to $\mathcal{S}$, for $\mathcal{S}'$ to be a LETS structure, there must exist at least one cycle in the subgraph induced by $F' \setminus F$. With a discussion similar to the case of two connecting edges, one can prove that there must be exactly one cycle in the subgraph. For this case, the only possible configurations are the expansions of $\mathcal{S}$ by lollipop walks lo_m^c with $m \geq 3$ (length more than four) and $c \geq 3$ (cycles longer than three). ■

Based on Proposition 5, except the simple cycles, each prime structure of dot-based expansion can be generated by applying *path* or *lollipop* expansions to its largest LETS sub-structure. This sub-structure is either a simple cycle or not. If not, then Lemmas 3 and 4, and Proposition 5 show that we can obtain the sub-structure by applying a combination of the three expansion techniques to a simple cycle.

Theorem 1. *All LETS structures of variable-regular Tanner graphs for any variable degree d_v, and in any (a, b) class (that are not simple cycles), can be generated by applying a combination of depth-one tree (dot), path and lollipop expansions to simple cycles.*

Proof. The only LETS structures that are out of the reach of *dot* expansions are prime structures discussed in Chapter 3. These prime structures are either simple cycles or not. Based on Proposition 5, and the above discussions, the prime structures that are not simple cycles can be generated using the three expansion techniques starting from simple cycles. ■

Compared to the dot-based characterization, Theorem 1 provides a new characterization of LETS structures. Instead of using one expansion technique (*dot*) and

$$
\begin{array}{ll}
 & F_1 = \{f_2, f_3, f_9\} \quad \text{in (3,3) class} \\
\xrightarrow{dot} & F_2 = \{f_2, f_3, f_9, f_4\} \quad \text{in (4,2) class} \\
\xrightarrow{lo_3^3} & F_3 = \{f_2, f_3, f_9, f_4, f_5, f_{10}, f_6\} \quad \text{in (7,3) class} \\
\xrightarrow{dot} & F_4 = \{f_2, f_3, f_9, f_4, f_5, f_{10}, f_6, f_7\} \quad \text{in (8,2) class} \\
\xrightarrow{pa_2} & F_5 = \{f_2, f_3, f_9, f_4, f_5, f_{10}, f_6, f_7, f_8, f_1\} \quad \text{in (10,2) class} \\
\xrightarrow{dot} & F = \{f_1, \ldots, f_{11}\} \quad \text{in (11,1) class}
\end{array}
$$

Figure 5.3: Steps of generating the $(11, 1)$ LETS structure of Fig. 4.5 starting from s_3.

some prime structures (simple cycles plus some other prime structures) to characterize the LETS structures, the new characterization uses more expansion techniques (*dot*, *path* and *lollipop*) but has fewer prime structures (only simple cycles). In the general framework of Section 5.2, this corresponds to a forest with smaller number of trees but more variety of expansion types for internal links within each tree. This new characterization, however, does not provide us with a road-map to a more efficient search algorithm. The reason is that, if we continue to rely on the dot-based search algorithm as before, we still need to enumerate all the instances of the required prime structures as the input to the dot-based search algorithm. Having a characterization of the prime structures as expansions of simple cycles, although helpful in such an enumeration, has no effect on the required prime structures with relatively large a and b values, and the (huge) number of the instances of such structures that need to be enumerated.

To use the new expansions in the characterization such that they translate to a more efficient search algorithm, we need to use them for generating not only the prime structures but also the other LETS structures of interest. This corresponds to a full reorganization of the internal structure of each tree within the forest. If the hierarchical structure of the trees is properly designed, many of the prime structures may not be needed in the new trees. This can, consequently, reduce the search complexity significantly. The following example explains this idea and motivates the *dpl* characterization of next section.

Example 8. *Consider the* $(11, 1)$ *LETS structure* $\mathcal{S}$ *of Fig. 4.5. The prime substructure of* $\mathcal{S}$ *with respect to dot expansion is* s_8*. There are, however, different ways to generate* $\mathcal{S}$ *starting from* s_3*, using a combination of dot, path and lollipop expansion techniques. One approach is explained in Fig. 5.3. In this figure,* F_i *is the set of nodes in* $\mathcal{S}_i$*, the LETS sub-structure of* $\mathcal{S}$ *in the* i*-th step of expansion. A search algorithm*

based on the sequence of expansions presented in Fig. 5.3, will start by enumerating all the instances of s_3 in the Tanner graph, instead of finding the instances of s_8, to find all the instances of $\mathcal{S}$.

As an example, consider the code discussed in Example 6. This code has 161 and 6,219,941 instances of s_3 and s_8, respectively. In dot-based search algorithm, all the instances of s_8 (more than 6 million) need to be enumerated and then dot *expansions need to be applied to each instance of s_8 three times successively to obtain the instances of $\mathcal{S}$. For this code, however, no instance of $\mathcal{S}$ exists in the Tanner graph. On the other hand, following the road-map of Fig. 5.3, by applying the* dot *expansion to the 161 instances of s_3, as the first step, one obtains no instance of the structure in the $(4, 2)$ class, and thus stops the search process for $\mathcal{S}$ (very quickly).*

In general, if all the LETS structures of interest, that have s_8 as their prime sub-structure in the dot *characterization, are generated using prime structures with smaller size, the instances of s_8 are no longer needed to be enumerated in the search algorithm.*

5.4 Dpl-based Characterization

Our goal in this section is to characterize all the non-isomorphic LETS structures of a variable-regular Tanner graph with variable degree d_{v} and girth g, within all the (a, b) classes with $a \leq a_{max}$ and $b \leq b_{max}$. The characterization is based on describing each LETS structure $\mathcal{S}$ as a hierarchy of embedded LETS structures that starts from a simple cycle, and expands to $\mathcal{S}$ in multiple steps, each step involving one of the three expansion techniques, i.e., *dot*, *path* or *lollipop*. The basic idea behind the new characterization, i.e., dpl-based characterization, is to generate as many LETS structures as possible within the interest range of a and b values by using smaller LETS structures that have a and b values also within the range. When translated to a search algorithm, this has the benefit that we will only search for instances of the structures that are themselves of direct interest to us (have a and b values within the target range). This is unlike the case for the dot-based search algorithm, where we would search for structures that are not of direct interest (have a or b values outside the target range), but happen to be parents of structures of interest in the *dot* characterization. To achieve the goal of staying within the target range of a and b values, and have an exhaustive characterization for all the LETS structures, we

need to use *path* and *lollipop* expansions in addition to the *dot* expansion. For the reasons just explained, one of the important properties of the *dpl* characterization is to generate a LETS structure in each level of expansion. With this constraint, i.e., all the sub-structures in the embedded sequence are LETSs, it is easy to see that the proposed characterization is *minimal*, in the sense that, none of the expansion steps can be divided into smaller expansions. We also prove that the three expansion techniques of *dot*, *path* and *lollipop*, are the only expansions that are needed in minimal characterizations. Within the framework of Section 5.2, in the design of the search forest, we aim at covering all the LETS structures of interest without introducing any auxiliary structures within the forest, if possible. This is along with the constraints that each tree in the forest is rooted at a short simple cycle and the connections within each tree correspond to only *dot*, *path* and *lollipop* expansions.

Given a target range of interest $a \leq a_{max}$ and $b \leq b_{max}$, we start from all the simple cycles within this range. Based on Proposition 2, for a given a, a simple cycle of size a has the largest b value of $a(d_{\text{v}} - 2)$ among all the (a, b) LETS structures. We thus initiate the characterization by including simple cycles $s_{g/2}, \ldots, s_{\lfloor b_{max}/(d_{\text{v}}-2)\rfloor}$. We then recursively apply *dot* expansions to these simple cycles to generate more LETS structures within the range of interest. (We choose the *dot* expansion in this step of the algorithm, since it can generate the majority of the LETS structures starting from the simple cycles.) If at the end of this process, there are still some LETS structures within the range of interest that are not generated, we try to generate them using the already generated smaller LETS structures by applying the *path* or *lollipop* expansions. Propositions 3 and 4, are our guide in this process. After the generation of any new LETS structure in this process, we also recursively apply *dot* expansions to it to (possibly) generate more new LETS structures in the range of interest. If there are still some structures that are not generated, we will increase the range of b values to $b_{max} + 1$ to include some new LETS structures (auxiliary structures) that can generate the missing LETS structures through expansions. This process of including LETS structures with larger b values will continue until all the LETS structures of all the (a, b) classes within the range $a \leq a_{max}$ and $b \leq b_{max}$, are generated.

When the process of generating LETS structures of interest, as described above, is completed, for each LETS structure $\mathcal{S}$ in the range, we have the parent simple cycle s_j, for some $j \geq g/2$, and the sequence of expansions that are applied to s_j to generate

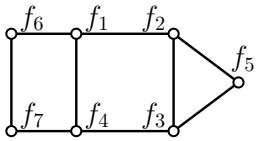


Figure 5.4: A LETS structure in the $(7,3)$ class of a variable-regular graph with $d_{\mathrm{v}} = 3$ and $g = 6$.

Minimal Characterization 1:

$F_1 = \{f_1, f_2, f_3, f_4\}$ in (4,4) class
$\xrightarrow{dot}$ $F_2 = \{f_1, f_2, f_3, f_4, f_5\}$ in (5,3) class
$\xrightarrow{pa_2}$ $F_3 = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$ in (7,3) class

Minimal Characterization 2:

$F_1 = \{f_1, f_2, f_3, f_4\}$ in (4,4) class
$\xrightarrow{pa_2}$ $F_2 = \{f_1, f_2, f_3, f_4, f_6, f_7\}$ in (6,4) class
$\xrightarrow{dot}$ $F_3 = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$ in (7,3) class

Figure 5.5: Two minimal characterizations of the $(7,3)$ LETS structure of Fig. 5.4, starting from s_4.

$\mathcal{S}$. We however, recognize that this minimal characterization of $\mathcal{S}$ may not be unique, in the sense that, there may be other combinations of parent and expansion sequences that result in the generation of $\mathcal{S}$. This is explained in the following example.

Example 9. *Fig. 5.4 shows a* $(7,3)$ *LETS structure* $\mathcal{S} = (\{f_1, f_2, \ldots, f_7\}, E)$ *of a variable-regular graph with* $d_{\mathrm{v}} = 3$ *and* $g = 6$. *Two minimal characterizations of* $\mathcal{S}$ *starting from the simple cycle* s_4 *in the* $(4,4)$ *class* ($\mathcal{S}' = (\{f_1, f_2, \ldots, f_4\}, E')$) *are presented in Fig. 5.5.*

To present the *dpl* characterization in a compact form, instead of providing the above sequence of parent/expansions for each LETS structure, for each (a, b) class of interest, we only provide (in a characterization table) the dpl-prime structures (simple cycles) that are required to obtain the LETS structures in that class, plus the expansions that need to be applied to the structures within that class to obtain larger LETS structures within the range of interest. This presentation of characterization is particularly helpful for the dpl-based search algorithm.

The pseudo-code of the *dpl* characterization algorithm is given in Algorithm 1. In this algorithm, the table $\mathcal{EX}$ is used to store the sets of required expansion techniques

in different classes. The notation $\mathcal{EX}_{(a,b)}$ is used to denote the expansion sets for the (a, b) class. For example, $\mathcal{EX}_{(a,b)} = \{dot, pa_2, lo_3^3\}$ means that dot, pa_2 and lo_3^3 expansions will need to be applied to all the non-isomorphic LETS structures in the (a, b) class.

Algorithm 1 (Dpl-based Characterization Algorithm) Finds all the non-isomorphic LETS structures in (a, b) classes with $a \leq a_{max}$ and $b \leq b_{max}$, for variable-regular Tanner graphs with variable degree $d_{\rm v}$ and girth g. The output parameter K is the size of the largest simple cycle required in the characterization. The LETS structures generated from the simple cycle s_k are stored in $\mathcal{L}_k$, for $k = g/2, \ldots, K$, and the set of expansions for different classes are stored in $\mathcal{EX}$.

Inputs: a_{max}, b_{max}, $d_{\rm v}$ and g.
Initializations: $\mathcal{L} \leftarrow \emptyset$, $a = g/2$, $b'_{max} = b_{max}$, $\mathcal{EX}_{(a,b)} \leftarrow$dot, $\forall a \in \{g/2, \ldots, a_{max} - 1\}$, $\forall b \in \{2, \ldots, b'_{max}\}$.
$K = \lfloor b_{max}/(d_{\rm v} - 2) \rfloor$.
for $k = g/2, \ldots, K$ **do**
    $\mathcal{L}_k =$ **DotExpansion**$(s_k, \mathcal{L}, a_{max})$.
    $\mathcal{L} = \mathcal{L} \cup \mathcal{L}_k$.
end for
while $a \leq a_{max}$ **do**
    if all the LETS structures in classes of size a are not found (check by comparing the structures of size a in $\mathcal{L}$ with those generated by *geng*) **do**
        for any incomplete (a, b) class, where $1 \leq b \leq b_{max}$ **do**
            for $k = g/2, \ldots, K$ **do**
                $(\mathcal{L}_{tem}, \mathcal{EX}^{tem}) =$**PaLoExpan**$(a, b, \mathcal{L}_k, \mathcal{L})$.
                $\mathcal{L}'_{tem} =$ **DotExpansion**$(\mathcal{L}_{tem}, \mathcal{L}, a_{max})$.
                $\mathcal{L} = \mathcal{L} \cup \mathcal{L}'_{tem}$, $\mathcal{L}_k = \mathcal{L}_k \cup \mathcal{L}'_{tem}$.
                $\mathcal{EX} = \mathcal{EX} \cup \mathcal{EX}^{tem}$.
            end for
        end for
    end if
    $a = a + 1$.
end while

The algorithm presented in Routine 3 is responsible for *dot* expansion. In this routine, set $\mathcal{L}_k^a$ is the set of non-isomorphic LETS structures of size a generated by the recursive application of the *dot* expansion starting from the structures in $\mathcal{P}_k$. All the non-isomorphic LETS structures of size $a+1$ that are generated by expanding the non-isomorphic LETS structures in $\mathcal{L}_k^a$ using dot_m expansions with different values of m

21: **while** all the LETS structures in all the classes of interest are not found (check by comparing all the structures in $\mathcal{L}$ with those generated by *geng*) **do**
22: $b'_{max} = b'_{max} + 1$, $a = g/2$.
23: **while** $a < a_{max}$ **do**
24: $\mathcal{EX}_{(a,b'_{max})} \leftarrow dot$.
25: **if** (a, b'_{max}) is the class of a simple cycle **then**
26: $K = K + 1$.
27: $\mathcal{L}_K$= **DotExpansion**$(s_K, \mathcal{L}, a_{max})$.
28: $\mathcal{L} = \mathcal{L} \cup \mathcal{L}_K$.
29: **if** $\mathcal{L}_K = \emptyset$ **then**
30: $\mathcal{EX}_{(a,b'_{max})} = \mathcal{EX}_{(a,b'_{max})} \setminus dot$.
31: **end if**
32: **else**
33: **for** $k = g/2, \ldots, K$ **do**
34: $(\mathcal{L}_{tem}, \mathcal{EX}^{tem})$=**PaLoExpan**$(a, b'_{max}, \mathcal{L}_k, \mathcal{L})$.
35: $\mathcal{L}'_{tem}$= **DotExpansion**$(\mathcal{L}_{tem}, \mathcal{L}, a_{max})$.
36: $\mathcal{L} = \mathcal{L} \cup \mathcal{L}'_{tem}$, $\mathcal{L}_k = \mathcal{L}_k \cup \mathcal{L}'_{tem}$.
37: $\mathcal{EX} = \mathcal{EX} \cup \mathcal{EX}^{tem}$.
38: **end for**
39: **end if**
40: $a = a + 1$.
41: **end while**
42: **end while**
43: **Outputs:** K, $\{\mathcal{L}_{g/2}, \ldots, \mathcal{L}_K\}$, $\mathcal{EX}$.

are stored in $\mathcal{L}_{tem}$. A subset of $\mathcal{L}_{tem}$ that is not generated by smaller prime structures will then be identified as the set $\mathcal{L}_k^{a+1}$. The algorithm of Routine 4 is responsible for

Routine 3 (DotExpansion) Recursive expansion of a set of structures $\mathcal{P}_k$ of size k, up to size a_{max}, using *dot* expansion, excluding the already found structures $\mathcal{L}$, and storing the rest in $\mathcal{L}_k$. $\mathcal{L}_k$= DotExpansion $(\mathcal{P}_k, \mathcal{L}, a_{max})$

Initialization: $\mathcal{L}_k^k \leftarrow \mathcal{P}_k$.
for $a = k, \ldots, a_{max} - 1$ **do**
  $\mathcal{L}_{tem} \leftarrow \emptyset$.
  for each LETS structure $\mathcal{S}$ in $\mathcal{L}_k^a$ **do**
    Find the subset F' of nodes in $\mathcal{S}$ with degree strictly less than d_{v}.
    for $m = 2, \ldots, d_{\mathrm{v}}$ **do**
      Construct $\mathcal{N}_m(\mathcal{S})$ to include all the possible subsets of F' with size m.
      for each set of m nodes in $\mathcal{N}_m(\mathcal{S})$ **do**
        Generate a new structure, $\mathcal{S}'$ using dot_m expansion by connecting a new node to the set of m nodes.
        $\mathcal{L}_{tem} = \mathcal{L}_{tem} \cup \mathcal{S}'$.
      end for
    end for
  end for
  $\mathcal{L}_k^{a+1} = \mathcal{L}_{tem} \setminus \mathcal{L}$.
end for
$\mathcal{L}_k = \bigcup_{a=k}^{a_{max}} \mathcal{L}_k^a$.
Output: $\mathcal{L}_k$.

generating LETS structures that are out of the reach of *dot* expansions, by using *path* and *lollipop* expansions.

The following lemma proves that minimal characterizations are based only on the three expansions *dot*, *path* and *lollipop*.

Lemma 6. *Consider a characterization of a LETS structure $\mathcal{S}$, based on a hierarchy of embedded LETS structures that starts from a simple cycle and generates $\mathcal{S}$ through a series of graph expansions. Assume that the characterization is minimal, in the sense that, none of the expansions can be broken into a sequence of smaller expansions such that the resulting sub-structures are still LETSs. Then, any graph expansion in the series, corresponding to a minimal characterization, is either a dot, a path or a lollipop expansion.*

Routine 4 (PaLoExpansion) Finding new LETS structures in the (a, b) class which are out of the reach of *dot* expansion by applying *path* and *lollipop* expansions to LETS structures already found in $\mathcal{L}_k$, excluding the already found structures $\mathcal{L}$, and storing all the new structures in $\mathcal{L}_{tem}$. The expansions corresponding to the new structures are stored in $\mathcal{EX}^{tem}$. $(\mathcal{L}_{tem}, \mathcal{EX}^{tem})$= PaLoExpansion $(a, b, \mathcal{L}_k, \mathcal{L})$

Initialization: $\mathcal{EX}^{tem} \leftarrow \emptyset$, $\mathcal{L}_{tem} \leftarrow \emptyset$.
$a' = a - 2$.
while $a' \geq g/2$ **do**
    $m = a - a'$.
    $\mathcal{L}^p_{tem} \leftarrow \emptyset$, $\mathcal{L}_{c,tem} \leftarrow \emptyset, 3 \leq c \leq m$.
    for any LETS structure $\mathcal{S}$ in the $(a', b' = b + 2 - m(d_v - 2))$ class of $\mathcal{L}_k$ **do**
        $\mathcal{L}^p = \textbf{PathExpansion}(\mathcal{S}, m)$.
        $\mathcal{L}^p_{tem} = \mathcal{L}^p_{tem} \cup \mathcal{L}^p$.
        for any possible c **do**
            $\mathcal{L}^l_c = \textbf{LollipopExpansion}(\mathcal{S}, m, c)$.
            $\mathcal{L}_{c,tem} = \mathcal{L}_{c,tem} \cup \mathcal{L}^l_c$.
        end for
    end for
    if $\{\mathcal{L}^p_{tem} \setminus \mathcal{L}\} \neq \emptyset$ and $\{\mathcal{L}^p_{tem} \setminus \mathcal{L}_{tem}\} \neq \emptyset$ **then**
        $\mathcal{L}_{tem} = \mathcal{L}_{tem} \cup \{\mathcal{L}^p_{tem} \setminus \mathcal{L}\}$.
        $\mathcal{EX}^{tem}_{(a',b')} \leftarrow \mathcal{EX}^{tem}_{(a',b')} \cup pa_m$.
    end if
    for any possible c **do**
        if $\{\mathcal{L}_{c,tem} \setminus \mathcal{L}\} \neq \emptyset$ and $\{\mathcal{L}_{c,tem} \setminus \mathcal{L}_{tem}\} \neq \emptyset$ **then**
            $\mathcal{L}_{tem} = \mathcal{L}_{tem} \cup \{\mathcal{L}_{c,tem} \setminus \mathcal{L}\}$.
            $\mathcal{EX}^{tem}_{(a',b')} \leftarrow \mathcal{EX}^{tem}_{(a',b')} \cup lo^c_m$.
        end if
    end for
    $a' = a' - 1$.
end while
Outputs: $\mathcal{L}_{tem}, \mathcal{EX}^{tem}$.

Proof. Consider structures $\mathcal{S}' = (F', E')$ and $\mathcal{S}'' = (F'', E'')$ as two successive embedded LETS structures in a minimal characterization of $\mathcal{S}$ ($\mathcal{S}''$ is generated by applying a graph expansion to $\mathcal{S}'$). If $|F'| = |F''| - 1$, then it is easy to see that $\mathcal{S}''$ is obtained from $\mathcal{S}'$ by a *dot* expansion. For the case of $|F'| = |F''| - 2$, using discussions similar to those in the proof of Case (1) of Proposition 5, one can prove that the only possible expansions to generate $\mathcal{S}''$ from $\mathcal{S}'$ are the closed and open paths of length 3. Finally, for the case of $|F'| < |F''| - 2$, following similar steps as those in the proof of Case (2) of Proposition 5, we can show that the only possible expansions to generate $\mathcal{S}''$ from $\mathcal{S}'$ are closed and open paths of length more than 3, or lollipop walks lo_m^c with $m \geq 3$ (length more than four) and $c \geq 3$ (cycles longer than three). ■

Algorithm 1 obtains minimal characterizations for LETS structures. The following theorem shows that the algorithm performs this task efficiently and effectively, by making sure that all the intermediate LETS structures generated in each characterization are within the target range of a and b values, if possible.

Theorem 2. *For given values of d_v, g, a_{max} and b_{max}, Algorithm 1 provides an optimal minimal characterization of all non-isomorphic (a, b) LETS structures of variable-regular graphs with variable degree d_v and girth g, with $a \leq a_{max}$ and $b \leq b_{max}$, in the sense that, to generate all such structures starting from simple cycles and using dot, path, and lollipop expansions, the maximum length K of the required simple cycles, and the maximum range b'_{max} of the b values of the LETS structures that are generated in the characterization process, are minimized by Algorithm 1.*

Proof. Assume that Algorithm 1 fails to characterize all the non-isomorphic (a, b) LETS structures with $a \leq a_{max}$ and $b \leq b_{max}$, in Lines 4–20 of the algorithm. Consider one such structure $\mathcal{S}$. The failure of Algorithm 1 to characterize $\mathcal{S}$ implies, based on Lemma 6 and Propositions 1, 3 and 4, that there is no minimal characterization of $\mathcal{S}$ that starts from a simple cycle in the range of interest with all the sub-structures also in the range. To obtain a minimal characterization for $\mathcal{S}$, one has thus no choice other than increasing the range of b values beyond b_{max}. In Lines 21-42, Algorithm 1 performs this task step by step, increasing the range of b values only by one, at each step, ensuring that the range increase is the minimum required to obtain a minimal characterization for all the LETS structures in the range. ■

Table 5.2: Information about some examples of characterization tables for different values of d_v, g, a_{max} and b_{max}

d_v	3	3	3	3	3	3	4	4	4	4	5
g	6	6	6	6	8	8	6	6	6	8	6
a_{max}	6	8	10	12	10	12	6	8	10	11	9
b_{max}	3	3	3	5	4	4	4	6	10	10	11
$b'_{max}(dpl)$	4	4	5	6	5	6	6	8	12	12	13
Cycle Primes	s_3, s_4	s_3, s_4	s_3, s_4, s_5	s_3, s_4, s_5, s_6	s_4, s_5	s_4, s_5, s_6	s_3	s_3, s_4	s_3, s_4, s_5	s_3, s_4	s_3, s_4
Expansion Techniques	dot	dot	dot, pa_2	dot, pa_2, pa_3	dot, pa_2	dot, pa_2	dot	dot	dot, pa_2, pa_3	dot, pa_2	dot
	pa_2	pa_2	pa_3	pa_4, pa_5	pa_3	pa_3, pa_4		pa_2	pa_4, pa_5, lo_3^3	pa_3	pa_2
		pa_3	lo_3^3, lo_4^3	$lo_3^3, lo_4^3, lo_4^4, lo_5^3$	pa_4	lo_4^4		pa_3	lo_4^3, lo_4^4, lo_5^3	pa_4	pa_3
		lo_3^3	lo_4^4	$lo_6^3, lo_6^4, lo_6^5, lo_6^6$	lo_4^4	lo_5^4, lo_5^5		lo_3^3	lo_5^4, lo_5^5	lo_4^4	lo_3^3
Rate†-Length‡	h-s	h-s,h-m	m-s,m-m	m-m,m-l	h-l,m-s	m-m,m-l	h-s	h-m,h-l	m-m,m-l	m-m,m-l	h-l,m-s
	h-m	h-l,m-s		l-s,l-m		l-s		m-s	l-s	l-s	m-m
$b'_{max}(dot)$	5	6	8	9	8	9	6	12	16	16	18

† For Rate: l = low ($R < 0.3$), m = medium ($0.3 < R < 0.7$), h = high ($R > 0.7$).
‡ For Length: s = short ($n < 2000$), m = medium ($2000 < n < 8000$), l = large ($n > 8000$).

5.5 Dpl-based Characterization Tables

Given the values of d_v and g, and a target range $a \leq a_{max}$ and $b \leq b_{max}$, Algorithm 1 can be used to characterize all the LETS structures of variable-regular Tanner graphs in (a, b) classes of interest. For a given graph (code), the target values a_{max} and b_{max}, however, need to be selected such that the classes that have the main contribution in the error floor of the code, the so-called *dominant* classes, are included in the range. The proper selection of the range would depend on the code's rate and block length. Some information about the characterization tables for different values of d_v, g, a_{max} and b_{max} is provided in Table 5.2.

Row five of Table 5.2 contains the maximum b value of the LETS structures that need to be generated for the exhaustive coverage of the classes of interest. This corresponds to b'_{max} in Algorithm 1. This row should be compared to the last row of the table that shows the similar parameter when *dot* characterization is used. The comparison of the two rows demonstrates the advantage of the dpl-based approach compared to the dot-based approach in covering the same range of LETS classes but by generating LETS structures with smaller b values.

Example 10. *In dot-based (LSS-based) approach, to find all the LETS structures of variable regular graphs with $d_v = 3$ and $g = 6$, in the interest range of $a \leq 10$ and $b \leq 3$, all the LETS structures within the range $a \leq 8$, $b \leq 8$, need to be generated.*

However, in the proposed approach, only the LETS structures with $b \leq 5$ need to be generated to guarantee the generation of all the LETS structures in the interest range. This corresponds to Characterization Table 5.5.

Rows six and seven of Table 5.2 contain the simple cycles and the expansion techniques that are used in each characterization table, respectively.

Example 11. *Consider the case discussed in Example 10. Table 5.2 shows that for this case, to characterize all the LETS structures in the range of interest, simple cycles of length 3, 4 and 5, and expansions $dot, pa_2, pa_3, lo_3^3, lo_4^3$ and lo_4^4 are needed.*

We note that while the study of the relative harmfulness of different LETS structures is beyond the scope of this work, our experimental results, as well as those reported in the literature, show that the dominant LETS structures/classes depend not only on d_{v} and g, but also on the rate R and block length n of the code. As a result, the values of interest for a_{max} and b_{max} would also depend on all these parameters (d_{v}, g, R and n). In the eighth row of Table 5.2, we have indicated the ranges of rates and block lengths for which a characterization table can be useful, i.e., for a code within the specified range of rate and block length, the table is likely to cover the dominant classes of LETSs. Since the characterization tables are used as guidelines for dpl-based search algorithm to find instances of LETS structures in a given code (graph), one is interested in selecting the smallest values for a_{max} and b_{max} to minimize the search complexity while ensuring that the dominant LETS structures are covered (found) in the search process. While this has been the main motivation for providing characterization tables, we make no claim that these tables are necessarily optimal in the sense just explained, nor we assert that such universally optimal tables even exist for codes with the same d_{v}, g, rate and block length. These tables should thus be only treated as suggestions rather than definite guidelines.

Example 12. *Consider the Characterization Table 5.5. The information in Table 5.2 indicates that Table 5.5 is likely to contain the dominant LETS classes of medium-rate codes ($0.3 < R < 0.7$) with short to medium block length ($n < 8000$), with $d_{\text{v}} = 3$, and $g = 6$.*

In each characterization table, columns and rows correspond to different values of a and b, respectively. For each (a, b) class of LETSs, the top entries in the table

Table 5.3: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_{\mathrm{v}} = 3$ and $g = 6$ for $a \leq a_{max} = 6$ and $b \leq b_{max} = 3$

	$a = 3$	$a = 4$	$a = 5$	$a = 6$
b = 0	-	$s_3(1)$ – – – –	-	$s_4(2)$ – – – –
b = 1	-	-	$s_3(1)$ – – – –	-
b = 2	-	$s_3(1)$ – – – dot, pa_2	-	$s_3(1), s_4(3)$ – – – –
b = 3	$\boldsymbol{s_3(1)}$ – – – dot	-	$s_4(2)$ – – – dot	-
b = 4	-	$\boldsymbol{s_4(1)}$ – – – dot	-	– – – –

Table 5.4: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_{\mathrm{v}} = 3$ and $g = 6$ for $a \leq a_{max} = 8$ and $b \leq b_{max} = 3$

	$a = 3$	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$
b = 0	-	$s_3(1)$ – – – –	-	$s_4(2)$ – – – –	-	$s_3(3), s_4(2)$ – – – –
b = 1	-	-	$s_3(1)$ – – – lo_3^3	-	$s_3(3), s_4(1)$ – – – –	-
b = 2	-	$s_3(1)$ – – – dot, pa_2, pa_3	-	$s_3(1), s_4(3)$ – – – dot, pa_2	-	$s_3(11), s_4(8)$ – – – –
b = 3	$\boldsymbol{s_3(1)}$ – – – dot, pa_3, lo_3^3	-	$s_4(2)$ – – – dot	-	$s_3(6), s_4(4)$ – – – dot	-
b = 4	-	$\boldsymbol{s_4(1)}$ – – – dot, pa_2	-	– – – dot	-	– – – –

Table 5.5: Characterization (cycle prime sub-graphs and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_{\mathrm{v}} = 3$ and $g = 6$ for $a \leq a_{max} = 10$ and $b \leq b_{max} = 3$

	$a = 3$	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$	$a = 9$	$a = 10$
b = 0	-	$s_3(1)$ – – – –	-	$s_4(2)$ – – – –	-	$s_3(3), s_4(2)$ – – – –	-	$s_3(13), s_4(5), s_5(1)$ – – – –
b = 1	-	-	$s_3(1)$ – – – lo_4^3, lo_4^4	-	$s_3(3), s_4(1)$ – – – lo_3^3	-	$s_3(15), s_4(4)$ – – – –	-
b = 2	-	$s_3(1)$ – – – dot, pa_2, pa_3	-	$s_3(1), s_4(3)$ – – – dot, pa_2, pa_3	-	$s_3(12), s_4(7)$ – – – dot, pa_2	-	$s_3(75), s_4(36), s_5(2)$ – – – –
b = 3	$\boldsymbol{s_3(1)}$ – – – dot, pa_3 lo_3^3, lo_4^3, lo_4^4	-	$s_4(2)$ – – – dot, pa_3	-	$s_3(6), s_4(4)$ – – – dot, pa_2	-	$s_3(40)$ $s_4(21), s_5(2)$ – – – dot	-
b = 4	-	$\boldsymbol{s_4(1)}$ – – – dot, pa_2	-	– – – dot, pa_2	-	– – – dot	-	– – – –
b = 5	-	-	$\boldsymbol{s_5(1)}$ – – – pa_2	-	– – – dot	-	– – – –	-

(separated by a dashed line from the bottom entries) show the prime simple cycles that are parents of the LETS structures within that class, and the multiplicity of non-isomorphic structures in the class with those parents (multiplicity is given within brackets). The bottom entries show the expansion techniques applied to all the LETS structures within the class. For example, $s_i(x), s_j(y)$, as upper entries in a class means that there are $x + y$ non-isomorphic LETS structures in that class, x number of such structures are generated from s_i and y of them are generated from s_j. Having dot, pa_m, lo^c_m, as lower entries in a class means that all the possible dot_m expansions, both closed and open *path* expansions (pa^o_m, pa^c_m), and lo^c_m are used to expand all the LETS structures of that class. Having the symbol "-", as the only entry for a class means that it is impossible to have LETS structures in that class. Having the symbol "-" as the only bottom entry means that no expansion technique is applied to the structures in that class. In each characterization table, the simple prime cycles that are needed to generate all the structures within the table are boldfaced.

Example 13. *Consider the LETS class* $(8, 4)$ *of Table 5.8. There are 10 non-isomorphic LETS structures in this class. Nine and one of these structures are generated starting from* s_4 *and* s_5*, respectively. To obtain all the LETS structures in the table, one needs to apply dot and* pa_2 *expansions to each one of the 10 structures in this class. This results in the generation of LETS structures in* $(9, 1), (9, 3)$ *and* $(10, 4)$ *classes (based on Propositions 1 and 3).*

As discussed in Section 5.4, to cover a given range of a and b values exhaustively, it is sometimes required to generate LETS structures with b values larger than b_{max} and up to some $b'_{max} > b_{max}$. In the characterization tables, the LETS classes with $b > b_{max}$ are separated from the rest by a double-line. For classes with $b > b_{max}$, the multiplicity of non-isomorphic LETS structures are not reported in the tables.

Remark 3. *Note that in Lines 21-42 of Algorithm 1, if an expansion technique does not generate a new LETS structure within the range of interest, it will not be added to* $\mathcal{EX}$*. For example, consider the characterization of the non-isomorphic LETS structures of* (a, b) *classes for variable-regular graphs with* $d_v = 3$ *and* $g = 6$*, in the range* $a \leq a_{max} = 10$ *and* $b \leq b_{max} = 3$ *(Table 5.5). In Line 34 of Algorithm 1,* pa_4 *is applied to the simple cycle* s_3 *in the* $(3, 3)$ *class and generates one new LETS structure in the* $(7, 5)$ *class. However, the new LETS structure in the* $(7, 5)$ *class does not*

Table 5.6: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_{\mathrm{v}} = 3$ and $g = 6$ for $a \leq a_{max} = 12$ and $b \leq b_{max} = 5$

	$a = 3$	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$	$a = 9$	$a = 10$	$a = 11$	$a = 12$
b = 0	-	$s_3(1)$ – – – –	-	$s_4(2)$ – – – –	-	$s_3(3)$ $s_4(2)$ – – – –	-	$s_3(13)$ $s_4(5)$ $s_5(1)$ – – – –	-	$s_3(63)$ $s_4(20)$ $s_5(2)$ – – – –
b = 1	-	-	$s_3(1)$ – – – lo_6^3, lo_6^4 lo_6^5, lo_6^6	-	$s_3(3), s_4(1)$ – – – lo_5^3, lo_5^4, lo_5^5	-	$s_3(15)$ $s_4(4)$ – – – –	-	$s_3(91)$ $s_4(22)$ $s_5(1)$ – – – –	-
b = 2	-	$s_3(1)$ – – – dot, pa_2, pa_3 pa_4, pa_5, lo_5^5 $lo_6^3, lo_6^4, lo_6^5, lo_6^6$	-	$s_3(1)$ $s_4(3)$ – – – dot, pa_4 lo_5^5, pa_5	-	$s_3(14)$ $s_4(5)$ – – – dot, pa_4 lo_4^3, lo_4^4	-	$s_3(85)$ $s_4(27)$ $s_5(1)$ – – – dot	-	$s_3(641)$ $s_4(184)$ $s_5(10)$ – – – –
b = 3	$\boldsymbol{s_3(1)}$ – – – dot, pa_2, pa_3 pa_4, pa_5, lo_3^3 $lo_4^3, lo_4^4, lo_5^3, lo_5^4$	-	$s_4(2)$ – – – dot lo_4^4, lo_5^4	-	$s_3(6), s_4(4)$ – – – dot, pa_4, lo_3^3 lo_4^3, lo_4^4	-	$s_3(44)$ $s_4(17)$ $s_5(2)$ – – – dot, pa_3, lo_3^3	-	$s_3(355)$ $s_4(120)$ $s_5(7)$ – – – dot	-
b = 4	-	$\boldsymbol{s_4(1)}$ – – – dot, pa_2 pa_3, pa_4	-	$s_3(2)$ $s_4(2)$ – – – dot, pa_2 pa_3, pa_4	-	$s_3(14)$ $s_4(10)$ $s_5(1)$ – – – dot pa_2, pa_3	-	$s_3(129)$ $s_4(63)$ $s_5(6)$ – – – dot, pa_2	-	$s_3(1315)$ $s_4(524)$ $s_5(52)$ $s_6(1)$ – – – –
b = 5	-	-	$\boldsymbol{s_5(1)}$ – – – dot, pa_2 pa_3	-	$s_3(3)$ $s_4(2), s_5(1)$ – – – dot, pa_2 pa_3, lo_3^3	-	$s_3(30)$ $s_4(18)$ $s_5(4)$ – – – dot, pa_2	-	$s_3(328)$ $s_4(180)$ $s_5(27), s_6(1)$ – – – dot	-
b = 6	-	-	-	$\boldsymbol{s_6(1)}$ – – – dot, pa_2	-	– – – dot, pa_2	-	– – – dot	-	– – – –

Table 5.7: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_v = 3$ and $g = 8$ for $a \leq a_{max} = 10$ and $b \leq b_{max} = 4$

	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$	$a = 9$	$a = 10$
b = 0	-	-	$s_4(1)$ – – – –	-	$s_4(2)$ – – – –	-	$s_4(5), s_5(1)$ – – – –
b = 1	-	-	-	$s_4(1)$ – – – –	-	$s_4(4)$ – – – –	-
b = 2	-	-	$s_4(1)$ – – – dot, pa_3, pa_4	-	$s_4(5)$ – – – dot	-	$s_4(27), s_5(1)$ – – – –
b = 3	-	$s_4(1)$ – – – dot, pa_4, lo_4^4	-	$s_4(3)$ – – – dot	-	$s_4(16), s_5(1)$ – – – dot	-
b = 4	$\boldsymbol{s_4(1)}$ – – – dot, pa_2, pa_3	-	$s_4(2)$ – – – dot, pa_2, pa_3	-	$s_4(9), s_5(1)$ – – – dot, pa_2	-	$s_4(57), s_5(6)$ – – – –
b = 5	-	$\boldsymbol{s_5(1)}$ – – – pa_2	-	– – – dot, pa_2	-	– – – dot	-

generate any new non-isomorphic LETS structure in the interest range of $a \leq 10$ *and* $b \leq 3$*. Therefore,* pa_4 *expansion can be removed from the list of expansion techniques applied to* s_3 *in the* $(3, 3)$ *class.*

Remark 4. *In Lines 21-42 of Algorithm 1, if there are still some LETS structures within the range of interest that are not generated, the range of* b *values will be increased to include some new LETS structures that can generate the missing LETS structures through expansions. We note that, in this process, some LETS structures, which are already generated in Lines 8-20 of Algorithm 1, may be regenerated. In such cases, some path or lollipop expansions, which were added in Lines 8-20 of Algorithm 1, could be removed. For example, consider the characterization of the non-isomorphic LETS structures of* (a, b) *classes for variable-regular graphs with* $d_v = 3$ *and* $g = 6$*, in the range* $a \leq a_{max} = 10$ *and* $b \leq b_{max} = 3$ *(Table 5.5). In Lines 8-20 of Algorithm 1,* pa_2 *is applied to the simple cycle* s_3 *in the* $(3, 3)$ *class and generates one new LETS structure in the* $(5, 3)$ *class. However, since there are still some structures missing by the time that the algorithm reaches Line 21, the range of* b *values will be increased to* $b'_{max} = 4$*. In Lines 21-42 of Algorithm 1, by applying the dot expansion to the simple*

Table 5.8: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_{\mathrm{v}} = 3$ and $g = 8$ for $a \leq a_{max} = 12$ and $b \leq b_{max} = 4$

	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$	$a = 9$	$a = 10$	$a = 11$	$a = 12$
b = 0	-	-	$s_4(1)$ – – – –	-	$s_4(2)$ – – – –	-	$s_4(5), s_5(1)$ – – – –	-	$s_4(20), s_5(2)$
b = 1	-	-	-	$s_4(1)$ – – – lo_5^4, lo_5^5	-	$s_4(4)$ – – – –	-	$s_4(22), s_5(1)$ – – – –	-
b = 2	-	-	$s_4(1)$ – – – dot, pa_3 pa_4	-	$s_4(5)$ – – – dot, pa_3 pa_4, lo_4^4	-	$s_4(27), s_5(1)$ – – – dot	-	$s_4(179)$ $s_5(11)$ – – – –
b = 3	-	$s_4(1)$ – – – dot, pa_4 lo_4^4, lo_5^4	-	$s_4(3)$ – – – dot, pa_4	-	$s_4(16)$ $s_5(1)$ – – – dot, pa_3	-	$s_4(115)$ $s_5(7)$ – – – dot	-
b = 4	$\boldsymbol{s_4(1)}$ – – – dot, pa_2 pa_3	-	$s_4(2)$ – – – dot, pa_2 pa_3, lo_4^4	-	$s_4(9), s_5(1)$ – – – dot, pa_2	-	$s_4(57), s_5(6)$ – – – dot, pa_2	-	$s_4(481)$ $s_5(48), s_6(10)$ – – – –
b = 5	-	$\boldsymbol{s_5(1)}$ – – – pa_2	-	– – – dot, pa_2, pa_3	-	– – – dot, pa_2	-	– – – dot	-
b = 6	-	-	$\boldsymbol{s_6(1)}$ – – – pa_2	-	– – – dot, pa_2	-	– – – dot	-	– – – –

cycle s_4, both of the non-isomorphic LETS structures in the $(5,3)$ class will be generated. Therefore, pa_2 expansion can be removed from the list of expansion techniques applied to s_3 in the $(3,3)$ class.

Table 5.9: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a,b) Classes for Variable-Regular Graphs with $d_v = 4$ and $g = 6$ for $a \leq a_{max} = 6$ and $b \leq b_{max} = 4$

	$a=3$	$a=4$	$a=5$	$a=6$
b = 0	-	-	$s_3(1)$ – – – –	$s_3(1)$ – – – –
b = 1	-	-	-	-
b = 2	-	-	$s_3(1)$ – – – *dot*	$s_3(3)$ – – – –
b = 3	-	-	-	-
b = 4	-	$s_3(1)$ – – – *dot*	$s_3(2)$ – – – *dot*	$s_3(7)$ – – – –
b = 5	-	-	-	-
b = 6	**$s_3(1)$** – – – *dot*	– – – *dot*	– – – *dot*	– – – –

Remark 5. *When Algorithm 1 will have to go beyond LETS structures with $b \leq b_{max}$, and to use simple cycles of size larger than $\lfloor b_{max}/(d_v - 2) \rfloor$, it may happen that the new cycle does not result in any new LETS structure in the range of interest. In such cases, the new cycle is not included as a prime structure in the table. An example of this can be seen in Table 5.11, where s_6 does not generate any new non-isomorphic LETS structure in the interest range of $a \leq 10$ and $b \leq 10$, and is thus not added as a prime structure in the table.*

Remark 6. *In characterization tables for the same values of d_v and g, but different ranges of a and b values, there may be some classes, for which, in different tables, different expansion techniques are used. For example, for the class of $(6,2)$ LETS structures in Tables 5.4 and 5.5, the expansions are different, and are respectively, $\{dot, pa_2\}$ and $\{dot, pa_2, pa_3\}$. Similarly, there may be cases where a LETS structure*

Table 5.10: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_{\mathrm{v}} = 4$ and $g = 6$ for $a \leq a_{max} = 8$ and $b \leq b_{max} = 6$

	$a = 3$	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$
b = 0	-	-	$s_3(1)$ – – – –	$s_3(1)$ – – – –	$s_3(2)$ – – – –	$s_3(5), s_4(1)$ – – – –
b = 1	-	-	-	-	-	-
b = 2	-	-	$s_3(1)$ – – – dot, pa_3	$s_3(3)$ – – – dot	$s_3(9)$ – – – dot	$s_3(34), s_4(1)$ – – – –
b = 3	-	-	-	-	-	-
b = 4	-	$s_3(1)$ – – – dot, lo_3^3	$s_3(2)$ – – – dot	$s_3(7)$ – – – dot, pa_2	$s_3(27), s_4(1)$ – – – dot	$s_3(122), s_4(2)$ – – – –
b = 5	-	-	-	-	-	-
b = 6	$\boldsymbol{s_3(1)}$ – – – dot	$s_3(1)$ – – – dot, pa_2	$s_3(3)$ – – – dot, pa_2	$s_3(10), s_4(1)$ – – – dot	$s_3(43), s_4(1)$ – – – dot	$s_3(226), s_4(5)$ – – – –
b = 7	-	-	-	-	-	-
b = 8	-	$\boldsymbol{s_4(1)}$ – – – dot	– – – dot	– – – dot	– – – dot	– – – –

Table 5.11: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_{\mathrm{v}} = 4$ and $g = 6$ for $a \leq a_{max} = 10$ and $b \leq b_{max} = 10$

	$a = 3$	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$	$a = 9$	$a = 10$
b = 0	-	-	$s_3(1)$ – – – –	$s_3(1)$ – – – –	$s_3(2)$ – – – –	$s_3(5), s_4(1)$ – – – –	$s_3(16)$ – – – –	$s_3(57), s_4(2)$ – – – –
b = 1	-	-	-	-	-	-	-	-
b = 2	-	-	$s_3(1)$ – – – dot, pa_5	$s_3(3)$ – – – dot	$s_3(9)$ – – – dot	$s_3(34), s_4(1)$ – – – dot	$s_3(152), s_4(2)$ – – – dot	$s_3(840), s_4(5)$ – – – –
b = 3	-	-	-	-	-	-	-	-
b = 4	-	$s_3(1)$ – – – dot, pa_4 lo_5^3, lo_5^4, lo_5^5	$s_3(2)$ – – – dot	$s_3(7)$ – – – dot, pa_3 pa_4, lo_4^4	$s_3(26)$ $s_4(1), s_5(1)$ – – – dot	$s_3(122)$ $s_4(2)$ – – – dot	$s_3(656)$ $s_4(7)$ – – – dot	$s_3(4140)$ $s_4(33)$ – – – –
b = 5	-	-	-	-	-	-	-	-
b = 6	$\boldsymbol{s_3(1)}$ – – – dot, pa_2, pa_3 lo_3^3, lo_4^3, lo_4^4	$s_3(1)$ – – – dot pa_3, pa_4	$s_3(3)$ – – – dot, pa_3 pa_4, lo_4^3	$s_3(10)$ $s_4(1)$ – – – dot, pa_2, pa_3	$s_3(42)$ $s_4(2)$ – – – dot, pa_3	$s_3(224)$ $s_4(7)$ – – – dot, pa_2	$s_3(1360)$ $s_4(19)$ – – – dot	$s_3(9382)$ $s_4(111)$ – – – –
b = 7	-	-	-	-	-	-	-	-
b = 8	-	$\boldsymbol{s_4(1)}$ – – – dot, pa_2	$s_3(2), s_4(1)$ – – – dot, pa_2 pa_3	$s_3(7)$ $s_4(2), s_5(1)$ – – – dot, pa_2	$s_3(39)$ $s_4(4), s_5(1)$ – – – dot, pa_2	$s_3(236)$ $s_4(14)$ – – – dot, pa_2	$s_3(1561)$ $s_4(52)$ – – – dot	$s_3(11719)$ $s_4(286)$ – – – –
b = 9	-	-	-	-	-	-	-	-
b = 10	-	-	$\boldsymbol{s_5(1)}$ – – – dot, pa_2	$s_3(3)$ $s_4(2)$ – – – dot, pa_2	$s_3(21)$ $s_4(6)$ – – – dot	$s_3(142)$ $s_4(19), s_5(1)$ – – – dot	$s_3(1060)$ $s_4(82), s_5(2)$ – – – dot	$s_3(8767)$ $s_4(476), s_5(1)$ – – – –
b = 11	-	-	-	-	-	-	-	-
b = 12	-	-	-	– – – –	– – – dot	– – – dot	– – – dot	– – – –

is generated starting from a different prime simple cycle. For example, comparison of the entries of the $(8, 2)$ *class in Tables 5.4 and 5.5 reveals that such LETS structures exist in this class.*

Table 5.12: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_v = 4$ and $g = 8$ for $a \leq a_{max} = 11$ and $b \leq b_{max} = 12$

	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$	$a = 9$	$a = 10$	$a = 11$
b = 0	-	-	-	-	$s_4(1)$ – – – –	-	$s_4(2)$ – – – –	$s_4(2)$ – – – –
b = 1	-	-	-	-	-	-	-	-
b = 2	-	-	-	-	$s_4(1)$ – – – *dot*	$s_4(2)$ – – – *dot*	$s_4(5)$ – – – *dot*	$s_4(19)$ – – – –
b = 3	-	-	-	-	-	-	-	-
b = 4	-	-	-	$s_4(1)$ – – – dot, pa_4, lo_4^4	$s_4(2)$ – – – *dot*	$s_4(7)$ – – – *dot*	$s_4(33)$ – – – *dot*	$s_4(164)$ – – – –
b = 5	-	-	-	-	-	-	-	-
b = 6	-	-	$s_4(1)$ – – – *dot*	$s_4(1)$ – – – dot, pa_3	$s_4(5)$ – – – dot, pa_3	$s_4(19)$ – – – dot, pa_2	$s_4(111)$ – – – *dot*	$s_4(706)$ – – – –
b = 7	-	-	-	-	-	-	-	-
b = 8	$\boldsymbol{s_4(1)}$ – – – dot, pa_2, pa_3	$s_4(1)$ – – – *dot*	$s_4(2)$ – – – dot, pa_2	$s_4(3)$ – – – *dot*	$s_4(14)$ – – – dot, pa_2	$s_4(50)$ – – – dot, pa_2	$s_4(286)$ – – – *dot*	$s_4(1902)$ – – – –
b = 9	-	-	-	-	-	-	-	-
b = 10	-	$\boldsymbol{s_5(1)}$ – – – pa_2	$s_4(2)$ – – – *dot*	$s_4(6)$ – – – *dot*	$s_4(19)$ – – – *dot*	$s_4(82)$ – – – *dot*	$s_4(475), s_5(1)$ – – – *dot*	$s_4(3223)$ – – – –
b = 11	-	-	-	-	-	-	-	-
b = 12	-	-	– – – –	– – – *dot*	– – – *dot*	– – – *dot*	– – – *dot*	– – – –

5.6 Dpl-based Search Algorithm

The characterization of LETS structures described in Sections 5.4 and 5.5 corresponds to an efficient search algorithm to find the instances of all (a, b) LETS structures of

Table 5.13: Characterization (cycle prime sub-structures and expansion techniques) of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-Regular Graphs with $d_{\mathrm{v}} = 5$ and $g = 6$ for $a \leq a_{max} = 9$ and $b \leq b_{max} = 11$

	$a=3$	$a=4$	$a=5$	$a=6$	$a=7$	$a=8$	$a=9$
b = 0	-	-	-	$s_3(1)$ – – – –	-	$s_3(3)$ – – – –	-
b = 1	-	-	-	-	$s_3(1)$ – – – –	-	$s_3(28)$ – – – –
b = 2	-	-	-	$s_3(1)$ – – – dot, pa_3	-	$s_3(16)$ – – – dot	-
b = 3	-	-	-	-	$s_3(6)$ – – – dot	-	$s_3(289)$ – – – –
b = 4	-	-	-	$s_3(2)$ – – – dot, pa_3, lo_3^3	-	$s_3(75)$ – – – dot	-
b = 5	-	-	$s_3(1)$ – – – dot, pa_3, lo_3^3	-	$s_3(18)$ – – – dot, pa_2	-	$s_3(1356), s_4(1)$ – – – –
b = 6	-	-	-	$s_3(5)$ – – – dot	-	$s_3(223)$ – – – dot	-
b = 7	-	-	$s_3(1)$ – – – dot, pa_2	-	$s_3(37)$ – – – dot, pa_2	-	$s_3(3786), s_4(1)$ – – – –
b = 8	-	$s_3(1)$ – – – dot	-	$s_3(8)$ – – – dot, pa_2	-	$s_3(460), s_4(1)$ – – – dot	-
b = 9	$\boldsymbol{s_3(1)}$ – – – dot, pa_2	-	$s_3(2)$ – – – dot, pa_2	-	$s_3(62)$ – – – dot	-	$s_3(7086), s_4(6)$ – – – –
b = 10	-	$s_3(1)$ – – – dot	-	$s_3(12)$ – – – dot	-	$s_3(691), s_4(2)$ – – – dot	-
b = 11	-	-	$s_3(3)$ – – – dot	-	$s_3(81), s_4(1)$ – – – dot	-	$s_3(9527), s_4(19)$ – – – –
b = 12	-	$\boldsymbol{s_4(1)}$ – – – dot	-	– – – dot	-	– – – dot	-
b = 13	-	-	– – – dot	-	– – – dot	-	– – – –

a variable-regular Tanner graph for a given range of a and b values in a guaranteed fashion. The efficiency of the search is implied by Theorem 2. The pseudo-code of the proposed search algorithm is given in Algorithm 2. The search algorithm starts by the enumeration of simple cycles of length up to K. These are the cycles that are identified in the characterization table as the required prime structures. For the sake of completeness, the pseudo-code for cycle enumeration is provided in Routine 9.

After the enumeration of all instances of a simple cycle, these instances are expanded in the while loop (Lines 10-24) to find instances of other LETS structures up to size a_{max}, $\mathcal{I}_k$. Notation $\mathcal{I}_k^a$ is used for the set of instances of size a found by starting from the instances of the simple cycle of length k, and $\mathcal{I}$ is the set of instances of all LETSs which are found so far in the algorithm. Finding the instances of LETS structures using *dot*, *path* and *lollipop* expansion techniques are explained in Routines 10, 11 and 12, respectively.

5.7 Complexity of the dpl-based Search Algorithm

The complexity of the search algorithm depends, in general, on the multiplicity of different instances of LETS structures and the expansion techniques used in different classes. The total complexity of the proposed algorithm can be divided into two parts: 1) complexity of finding the instances of prime simple cycle structures, and 2) complexity of expanding the instances of prime structures to find all the instances of every (a, b) LETS structure within the interest range of a and b values. In this section, we assume that the LDPC codes are regular.

5.7.1 Complexity of finding the instances of prime structures

The proposed dpl-based search algorithm uses Routine 9 to find the instances of simple cycles. Based on this approach, the order of complexity for finding the instances of a simple cycle of size k is $\mathcal{O}(n(d_{\mathrm{v}}d_{\mathrm{c}})^k)$. This complexity increases linearly, polynomially and exponentially with the block length, n, the check and variable degrees, $d_{\mathrm{c}}, d_{\mathrm{v}}$, and the size of the cycle, respectively. The complexity of finding the instances of a

Algorithm 2 (Dpl-based Search Algorithm) Finds all the instances of (a, b) LETS structures of a variable-regular Tanner graph G with girth g, for $a \leq a_{max}$ and $b \leq b_{max}$. The inputs are the maximum size K of the simple cycles, and the table $\mathcal{EX}$ of expansion techniques required in *dpl* characterization. The outputs are the sets $\mathcal{I}_k, k = g/2, \ldots, K$, which contain all the instances of LETS structures of size up to a_{max} that are expansions of simple cycles of length k in G.

Inputs: K, $\mathcal{EX}$, g, a_{max}, b_{max}.
Initializations: $\mathcal{I} \leftarrow \emptyset$.
for $k = g/2, \ldots, K$ **do**
    for $a = k, \ldots, a_{max}$ **do**
        $\mathcal{I}_k^a \leftarrow \emptyset$.
    end for
end for
for $k = g/2, \ldots, K$ **do**
    $\mathcal{I}_k^k$= **CycSrch**(s_k), $a = k$.
    while $a < a_{max}$ **do**
        $\mathcal{I}_{tem}^{a+1}$= **DotSrch**$(\mathcal{I}_k^a, \mathcal{I})$.
        for any pa_m in the characterization table **do**
            $\mathcal{I}_{tem}^{a+m}$=**PathSrch**$(\mathcal{I}_k^a, \mathcal{I}, \mathcal{EX}, m)$.
        end for
        for any lo_m^c in the characterization table **do**
            $\mathcal{I}'$=**LolliSrch**$(\mathcal{I}_k^a, \mathcal{I}_c^c, \mathcal{I}, \mathcal{EX}, m)$.
            $\mathcal{I}_{tem}^{a+m} = \mathcal{I}_{tem}^{a+m} \cup \mathcal{I}'$.
        end for
        for $t = a + 1, \ldots, a_{max}$ **do**
            $\mathcal{I}_k^t = \mathcal{I}_k^t \cup \mathcal{I}_{tem}^t$.
            $\mathcal{I} = \mathcal{I} \cup \mathcal{I}_{tem}^t$.
        end for
        $a = a + 1$.
    end while
    $\mathcal{I}_k = \bigcup_{a=k}^{a_{max}} \mathcal{I}_k^a$
end for
Outputs: $\{\mathcal{I}_{g/2}, \ldots, \mathcal{I}_K\}$.

Routine 5 (CycSrch) Finds all the instances $\mathcal{I}_k^k$ of simple cycles of length k in a given Tanner graph. $\mathcal{I}_k^k$= CycSrch(s_k)

Initializations: $\mathcal{I}_k^k \leftarrow \emptyset$.
for each variable node v_l in the Tanner graph **do**
  for each check node c_i in the neighborhood of v_l **do**
    Find all the paths $\mathcal{PA}_{i,l}$ of length $k-1$ in the Tanner graph, starting from c_i that do not contain v_l.
  end for
  for any pair of check nodes c_i and c_j in the neighborhood of v_l, where $i \neq j$ **do**
    for any path $pa \in \mathcal{PA}_{i,l}$, and any path $pa' \in \mathcal{PA}_{j,l}$ **do**
      if the two paths end with the same node and that node is their only common node **do**
        Let v and v' denote the last variable nodes of pa and pa', respectively.
        if variable nodes in $pa \setminus v$ and $pa' \setminus v'$ do not have any common check node **do**
          $\mathcal{S} = v_l \cup \{$set of variable nodes in $pa \cup pa'\}$.
          $\mathcal{I}_k^k = \mathcal{I}_k^k \cup \{\mathcal{S}\}$.
        end if
      end if
    end for
  end for
end for
Output: $\mathcal{I}_k^k$.

Routine 6 (DotSrch) Expansion of a set of instances $\mathcal{I}_k^a$ of LETS structures of size a using *dot* expansions to find a set of instances of LETS structures of size $a+1$, excluding the already found structures $\mathcal{I}$, and storing the rest in $\mathcal{I}'$. $\mathcal{I}'$= DotSrch ($\mathcal{I}_k^a, \mathcal{I}$)

Initializations: $\mathcal{I}_{tem} \leftarrow \emptyset$.
for each instance of LETS structure $\mathcal{S}$ in $\mathcal{I}_k^a$ **do**
  Consider $\mathcal{V}$ to be the set of variable nodes in $V \setminus \mathcal{S}$, which have at least two connections with the check nodes in $\Gamma_o(\mathcal{S})$ and have no connection with the check nodes in $\Gamma_e(\mathcal{S})$.
  for each variable node $v \in \mathcal{V}$ **do**
    $\mathcal{I}_{tem} = \mathcal{I}_{tem} \cup \{\mathcal{S} \cup v\}$.
  end for
end for
$\mathcal{I}' \leftarrow \mathcal{I}_{tem} \setminus \mathcal{I}$.
Output: $\mathcal{I}'$.

Routine 7 (PathSrch) Expanding the LETS instances of size a in $\mathcal{I}_k^a$ using pa_m to find instances of LETS structures of size $a+m$, excluding the already found instances $\mathcal{I}$, and storing the rest in $\mathcal{I}'$. Only instances in $\mathcal{I}_k^a$ whose structures in accordance to the expansion table $\mathcal{EX}$ need to be expanded by pa_m are expanded. $\mathcal{I}'$= PathSrch$(\mathcal{I}_k^a, \mathcal{I}, \mathcal{EX}, m)$

Initializations: $\mathcal{I}_{tem} \leftarrow \emptyset$.
for each LETS instance $\mathcal{S}$ in $\mathcal{I}_k^a$ that based on $\mathcal{EX}$ requires pa_m **do**
  for each unsatisfied check node $c_k \in \Gamma_o(\mathcal{S})$ **do**
    Find all the paths $\mathcal{PA}_k$ of length m in the Tanner graph, starting from c_k.
  end for
  for any pair of unsatisfied check nodes c_k and c_j in $\Gamma_o(\mathcal{S})$, where $k \neq j$ **do**
    for any path $pa \in \mathcal{PA}_k$ and any path $pa' \in \mathcal{PA}_j$ **do**
      if pa and pa' end with the same node and this node is their only common node, and if variable and check nodes of pa and pa' are not in $G(\mathcal{S})$ **do**
        $\mathcal{I}_{tem} \leftarrow \mathcal{I}_{tem} \cup \{\mathcal{S} \cup \{\text{set of variable nodes in } pa \cup pa'\}\}$.
      end if
    end for
  end for
end for
$\mathcal{I}' \leftarrow \mathcal{I}_{tem} \setminus \mathcal{I}$.
Output: $\mathcal{I}'$.

Routine 8 (LolliSrch) Expanding the LETS instances of size a in $\mathcal{I}_k^a$ using lo_m^c to find instances of LETS structures of size $a+m$, excluding the already found instances $\mathcal{I}$, and storing the rest in $\mathcal{I}'$. Only instances in $\mathcal{I}_k^a$ whose structures in accordance to the expansion table $\mathcal{EX}$ need to be expanded by lo_m^c are expanded. The set of instances of simple cycles of length c, $\mathcal{I}_c^c$, is also an input. $\mathcal{I}'$= LolliSrch $(\mathcal{I}_k^a, \mathcal{I}_c^c, \mathcal{I}, \mathcal{EX}, m)$

Initializations: $\mathcal{I}_{tem} \leftarrow \emptyset$, $d = m + 1 - c$.
for each LETS instance $\mathcal{S}$ in $\mathcal{I}_k^a$ that based on $\mathcal{EX}$ requires lo_m^c **do**
    Find all the paths $\mathcal{PA}$ of length $2(d-1)$ in the Tanner graph, starting from check nodes c' in $\Gamma_o(\mathcal{S})$ that have no common nodes with $G(\mathcal{S})$ other than c'.
    for each structure $\mathcal{S}' \in \mathcal{I}_c^c$, for which $G(\mathcal{S}')$ has no common node with $G(\mathcal{S})$, let $\Gamma_o(\mathcal{S}')$ denote the set of unsatisfied check nodes of $G(\mathcal{S}')$ and **do**
        for each path, $pa \in \mathcal{PA}$ **do**
            if pa ends with a check node c' in $\Gamma_o(\mathcal{S}')$ and if c' is the only common node between pa and $\Gamma_o(\mathcal{S}')$ **do**
                $\mathcal{I}_{tem} \leftarrow \mathcal{I}_{tem} \cup \{\mathcal{S} \cup \{\text{set of variable nodes in } pa \cup G(\mathcal{S}'\}\}$.
            end if
        end for
    end for
end for
$\mathcal{I}' \leftarrow \mathcal{I}_{tem} \setminus \mathcal{I}$.
Output: $\mathcal{I}'$.

simple cycle thus becomes quickly impractical as the cycle length increases.

Let N_k denote the multiplicity of the instances of the simple cycle s_k of length k in the Tanner graph. Clearly, the complexity of finding the instances of a LETS structure that is an immediate child of s_k is proportional to N_k. Moreover, all the instances of the prime cycles need to be stored for further processing throughout the search algorithm. The memory required to store all the instances of s_k is also proportional to N_k. To the best of our knowledge, there is no theoretical result on how N_k scales with n, k, or the degree distribution of the Tanner graph. Results in [9], however, suggest that N_k is rather independent of n and increases rather rapidly as the variable and check degrees, or k are increased. We note that, in comparison with the dot-based (LSS-based) search algorithm, the proposed dpl-based search algorithm requires smaller size prime cycles to be enumerated. This translates to less computational complexity and memory requirements as will be discussed in details in Section 5.8.

5.7.2 Complexity of expansions

Based on the characterization tables, a few expansion techniques (those in $\mathcal{EX}_{(a,b)}$) are used to expand the instances of LETS structures within each (a, b) class. The complexity of this part of the algorithm depends on the expansion techniques, and the multiplicity of LETSs in each (a, b) class, $N_{a,b}$. To the best of our knowledge, there is no theoretical result on how $N_{a,b}$ changes with a and b, or with different code parameters.

In the following, we discuss the complexity of the three expansion techniques. The expansion of an instance of a LETS structure in an (a, b) class using *dot* expansion is explained in Routine 10. For each instance, bd_{c} variable nodes should be checked to find the set $\mathcal{V}$ in Routine 10. The complexity for the expansion of all the instances of LETSs in an (a, b) class using *dot* is thus $\mathcal{O}(N_{a,b}b^2d_{\mathrm{c}}^2)$. The expansion of an instance of a LETS structure in an (a, b) class using the *path* expansion technique is explained in Routine 11. Based on the approach of Routine 11, it is easy to see that the complexity of expanding all the instance of LETS structures in an (a, b) class using pa_m is $\mathcal{O}(N_{a,b}b^2(d_{\mathrm{v}}d_{\mathrm{c}})^m)$. Similarly, the complexity of expanding all the instance of LETS structures in an (a, b) class using lo_m^c, based on Routine 12, is $\mathcal{O}(N_{a,b}N_cbc(d_{\mathrm{v}}d_{\mathrm{c}})^d)$, where N_c is the multiplicity of simple cycles of size c in the Tanner graph, and $d = m + 1 - c$.

Finally, we note that the memory required to store all the LETS instances of

Table 5.14: List of LDPC Codes Used in This Chapter

Code	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$	$\mathcal{C}_6$	$\mathcal{C}_7$	$\mathcal{C}_8$	$\mathcal{C}_9$	$\mathcal{C}_{10}$	$\mathcal{C}_{11}$	$\mathcal{C}_{12}$
n	169	361	529	1057	16383	504	816	1008	4000	20000	155	504
R	0.78	0.84	0.87	0.77	0.87	0.5	0.5	0.5	0.5	0.5	0.4	0.5
d_{v}	3	3	3	3	3	3	3	3	3	3	3	3
g	6	6	6	6	6	6	6	6	6	6	8	8
Ref.	[20]	[20]	[20]	[57]	[57]	[57]	[57]	[57]	[57]	[57]	[80]	[43]

Code	$\mathcal{C}_{13}$	$\mathcal{C}_{14}$	$\mathcal{C}_{15}$	$\mathcal{C}_{16}$	$\mathcal{C}_{17}$	$\mathcal{C}_{18}$	$\mathcal{C}_{19}$	$\mathcal{C}_{20}$	$\mathcal{C}_{21}$	$\mathcal{C}_{22}$	$\mathcal{C}_{23}$	$\mathcal{C}_{24}$
n	1008	2640	49	169	289	361	529	4095	16383	4000	8000	816
R	0.5	0.5	0.49	0.71	0.77	0.8	0.83	0.82	0.87	0.5	0.5	0.5
d_{v}	3	3	4	4	4	4	4	4	4	4	4	5
g	8	8	6	6	6	6	6	6	6	6	6	6
Ref.	[43]	[57]	[20]	[20]	[20]	[20]	[20]	[57]	[57]	[57]	[57]	[57]

an (a, b) class is $\mathcal{O}(aN_{a,b})$. The main advantage of using the dpl-based search algorithm, compared to dot-based search, is to avoid dealing with classes with large a and b values. Searching for LETS instances in these classes and expanding them imposes huge computational complexity and memory requirements on the search algorithm. Instead, in the dpl-based search algorithm, expansions are used in classes with relatively small $N_{a,b}$ values. Our experimental results in Section 5.8 show that the dpl-based search algorithm is significantly faster and requires much less memory compared to the *dot* (LSS)-based search algorithm, for different codes with a wide range of variable degrees, rates, block lengths and girths.

5.8 Numerical Results

We have applied the dot-based and dpl-based search algorithms to a large number of variable-regular random and structured LDPC codes with a wide range of variable degrees, rates and block lengths. These codes are listed in Table 6.5. For all the run-times reported in this work, a desktop computer with 8 processing cores (2.4-GHz CPU) and 8-GB memory is used. Except codes $\mathcal{C}_1$-$\mathcal{C}_3$, $\mathcal{C}_{11}$, $\mathcal{C}_{14}$ and $\mathcal{C}_{15}$-$\mathcal{C}_{19}$, which are structured codes, the other LDPC codes are all randomly constructed. For structured codes, their structure is not used to simplify the search, and all the runtimes are based on exhaustive search of the code's Tanner graph without taking into account the automorphisms that exist for structured codes.

Tables 5.15-5.25 list the multiplicity of instances of LETSs in different classes for

these codes, found by the dpl-based search algorithm, starting from the instances of simple cycles. Each row of a table corresponds to a LETS class, and for each class, the total number of instances of LETSs, EASs and FEASs are listed. We note that for the codes with $d_{\mathrm{v}} = 3$, the multiplicities of LETSs and EASs are the same. We have also listed the break down of the LETS multiplicity of each class based on the prime structure involved in the LETS structure. For example, in Table 5.15, for Code $\mathcal{C}_1$, and for the $(6, 2)$ class, we notice that there are a total of $142,974$ instances of LETS in this class, from which $54,756$ are children of s_3, and the remaining $88,218$ are children of s_4. The symbol "-" as an entry of an (a, b) class for a specific s_k means that, based on the characterization table, s_k is not a prime sub-structure of any LETS structure in the (a, b) class. In the last two rows of each table, the runtime of the dpl-based and the dot-based search algorithms is reported for comparison. The symbol "-" for the runtime of the dot-based search algorithm means that the search takes more than one day, or needs more than 8-GB memory to be completed. For fair comparison, both algorithms are run to find all the instances of (a, b) LETS structures within the range of interest for a and b values, exhaustively.

Table 5.15 lists the multiplicity of instances of LETSs and FEASs in all the nonempty classes within the range $a \leq 6$ and $b \leq 3$, for the codes $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$. These are high-rate array-based codes [20] with $d_{\mathrm{v}} = 3$ and $g = 6$. Absorbing and fully absorbing sets of these structured codes were studied in [17]. The search algorithm to find the instances of LETSs in the range $a \leq 6$ and $b \leq 3$, is based on the information provided in Table 5.3. Since the variable degree is 3, all the instances

Table 5.15: Multiplicities of LETS and FEAS structures of codes $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ within the range $a \leq 6$ and $b \leq 3$

	$\mathcal{C}_1$				$\mathcal{C}_2$				$\mathcal{C}_3$			
(a,b)	Primes		Total	Total	Primes		Total	Total	Primes		Total	Total
class	s_3	s_4	LETS	FEAS	s_3	s_4	LETS	FEAS	s_3	s_4	LETS	FEAS
(3,3)	2028	-	2028	0	6498	-	6498	0	11638	-	11638	0
(4,2)	3042	-	3042	3042	9747	-	9747	9747	17457	-	17457	17457
(5,3)	-	83148	83148	0	-	422370	422370	0	-	942678	942678	0
(6,0)	-	3718	3718	3718	-	18411	18411	18411	-	40733	40733	40733
(6,2)	54756	88218	142974	142974	292410	458109	750519	750519	663366	1029963	1693329	1693329
dpl	38 sec.				196 sec.				424 sec.			
dot	181 sec.				1447 sec.				5749 sec.			

of LETSs found by the search algorithm are EASs. Therefore, FEASs were found

Table 5.16: Multiplicities of LETS and FEAS structures of codes $\mathcal{C}_4$ and $\mathcal{C}_5$ within the range $a \leq 8$ and $b \leq 3$

	$\mathcal{C}_4$				$\mathcal{C}_5$			
(a,b)	Primes		Total	Total	Primes		Total	Total
class	s_3	s_4	LETS	FEAS	s_3	s_4	LETS	FEAS
(3,3)	2288	-	2288	1740	14291	-	14291	13491
(4,2)	296	-	296	267	404	-	404	394
(5,1)	29	-	29	29	10	-	10	10
(5,3)	-	17320	17320	12847	-	43904	43904	41382
(6,2)	650	2435	3085	2813	343	1284	1627	1590
(7,1)	246	23	269	269	34	3	37	37
(7,3)	146694	89023	235717	175759	156742	93724	250466	235926
(8,0)	16	0	16	16	0	0	0	0
(8,2)	25452	13658	39110	35524	5913	3084	8997	8824
dpl	3 min.				27 min.			
dot	-				-			

by examining the LETSs, and testing them for the definition of a fully elementary absorbing set. While it takes the proposed search algorithm 38, 196 and 424 seconds to find all the instances of LETS structures of Table 5.3 for these three codes, respectively, it took the dot-based search algorithm 181, 1447 and 5749 seconds to find the same set of LETSs for the three codes, respectively.

Table 5.16 lists the multiplicity of instances of LETSs and FEASs in all the nonempty classes within the range $a \leq 8$ and $b \leq 3$, for the codes $\mathcal{C}_4$ and $\mathcal{C}_5$. Both codes have $d_{\text{v}} = 3$ and $g = 6$. Code $\mathcal{C}_4$ is a high-rate code ($R = 0.77$) with short block length ($n = 1057$), and code $\mathcal{C}_5$ is a high-rate code ($R = 0.87$) with large block length ($n = 16,383$). The search to find the instances of LETSs in the range $a \leq 8$ and $b \leq 3$, is based on the information provided in Table 5.4. It takes the proposed search algorithm only about 3 and 27 minutes to find LETSs reported in this table for codes $\mathcal{C}_4$ and $\mathcal{C}_5$, respectively. To obtain the results of Table 5.4, the dpl-based search algorithm only needs to enumerate the instances of s_3 and s_4 in the Tanner graph of the codes. The dot-based search algorithm, on the other hand, needs to enumerate the instances of s_3, s_4, s_5, s_6, c_6, n_6, s_7, c_7, and n_7, to find all the instances of LETSs in the range $a \leq 8$ and $b \leq 3$, in a guaranteed fashion. It took the dot-based search algorithm 43 minutes to only find the 16,710,009 instances of s_5, and more than a day to find all the instances of LETSs reported in this table for code $\mathcal{C}_4$.

The examination of Table 5.16 shows that for two classes with the same size,

Table 5.17: Multiplicities of LETS and FEAS Structures of Codes $\mathcal{C}_6$, $\mathcal{C}_7$ and $\mathcal{C}_8$ within the range $a \leq 10$ and $b \leq 3$

	$\mathcal{C}_6$		$\mathcal{C}_7$		$\mathcal{C}_8$	
(a,b) class	Total LETS	Total FEAS	Total LETS	Total FEAS	Total LETS	Total FEAS
(3,3)	169	159	132	126	165	153
(4,2)	5	5	3	3	6	6
(5,3)	214	180	90	86	100	92
(6,2)	20	20	2	2	5	5
(7,3)	418	374	110	108	116	110
(8,2)	24	23	1	1	3	3
(9,1)	1	1	0	0	0	0
(9,3)	1127	992	195	166	169	161
(10,2)	71	69	15	15	4	4
dpl	20 sec.		18 sec.		20 sec.	
dot	948 sec.		918 sec.		942 sec.	

the class with a larger b value has more instances of LETSs. For example, for $\mathcal{C}_5$, the instances of LETS structures in the $(7,1)$ and $(7,3)$ classes are 37 and 250,466, respectively. Following a similar trend, this code probably has millions of instances of LETSs in the $(7,5)$ class, and it would take the search algorithm a long time to find them. These trapping sets, however, would not be dominant in the presence of those in the $(7,1)$ and $(7,3)$ classes. This highlights the importance of having a flexible and efficient search algorithm that can be easily tailored to different values of a_{max} and b_{max}.

Multiplicities of instances of LETSs and FEASs in all the nonempty classes within the range $a \leq 10$ and $b \leq 3$, for Codes $\mathcal{C}_6$, $\mathcal{C}_7$ and $\mathcal{C}_8$ are listed in Table 5.17. These are three $(3,6)$-regular LDPC codes, all with $g = 6$, and with short block lengths: 504, 816, and 1008, respectively. The search algorithm to find all the instances of LETSs in the range $a \leq 10$ and $b \leq 3$, is based on the information provided in Table 5.5. For each code, all the instances of LETSs and FEASs in this range are enumerated in less than 20 seconds. In comparison, the runtime for dot-based search is larger by a factor of about 45. It is worth mentioning that [97] only reports the multiplicity of FEAS structures in the $(3,3), (4,2)$ and $(5,3)$ classes of $\mathcal{C}_7$ and $\mathcal{C}_8$. The results of [97] match the results reported in the first three rows of Table 5.17.

Table 5.18 lists the multiplicity of instances of LETSs and FEASs in all the

Table 5.18: Multiplicities of LETS and FEAS structures of Codes $\mathcal{C}_7$ and $\mathcal{C}_8$ within the range $a \leq 12$ and $b \leq 5$

	$\mathcal{C}_7$							$\mathcal{C}_8$						
(a,b)	Primes				Total	Total	[53]	Primes				Total	Total	[53]
class	s_3	s_4	s_5	s_6	LETS	FEAS	FAS	s_3	s_4	s_5	s_6	LEAS	FEAS	FAS
(3,3)	132	-	-	-	132	126	126	165	-	-	-	165	153	153
(4,2)	3	-	-	-	3	3	3	6	-	-	-	6	6	6
(4,4)	-	1491	-	-	1491	1350	1350	-	1252	-	-	1252	1130	1130
(5,3)	-	90	-	-	90	86	86	-	100	-	-	100	**92**	**8388**
(5,5)	-	-	9169	-	9169	7286	7286	-	-	10019	-	10019	8388	8388
(6,2)	0	2	-	-	2	2	2	1	4	-	-	5	5	5
(6,4)	1084	1379	-	-	2463	**2235**	**2236**	1011	874	-	-	1885	**1666**	**1668**
(7,3)	54	56	-	-	110	108	108	74	42	-	-	116	**110**	**111**
(7,5)	13372	15957	4077	-	33406	26688	nr †	13462	12148	4126	-	29736	24876	24876
(8,2)	1	0	-	-	1	1	1	2	1	-	-	3	3	3
(8,4)	2145	2020	34	-	4199	3769	nr	1793	1136	32	-	2961	2617	nr
(9,3)	111	89	0	-	195	166	nr	122	47	-	-	169	161	nr
(9,5)	39306	39304	5768	-	84378	67558	nr	35996	23149	4642	-	63787	53596	nr
(10,2)	8	7	0	-	15	15	15	3	1	0	-	4	4	4
(10,4)	4389	3446	130	-	7965	7277	nr	3163	1609	97	-	4869	4421	nr
(11,3)	184	103	3	-	290	270	nr	154	64	1	-	219	209	nr
(11,5)	104491	95600	11066	116	211273	168214	nr	78596	47962	7619	59	134236	112213	nr
(12,2)	10	5	-	-	15	15	15	5	1	0	-	6	6	6
(12,4)	10163	7207	467	1	17838	15970	nr	6140	2669	231	1	9041	8168	nr
dpl	16 min.							14 min.						
dot	-							-						

† "nr" stands for *not reported*.

nonempty classes within the range $a \leq 12$ and $b \leq 5$, for Codes $\mathcal{C}_7$ and $\mathcal{C}_8$. It takes the proposed search algorithm 16 and 14 minutes to find all the instances of LETSs in the range of $a \leq 12$, $b \leq 5$, for Codes $\mathcal{C}_7$ and $\mathcal{C}_8$, respectively. This is versus about 20 seconds that takes the algorithm to find all the instances of LETSs of these codes in the range of $a \leq 10$, $b \leq 3$. In Table 5.18, we have also reported the multiplicity of instances of FASs obtained by the exhaustive search algorithm of [53]. There are only four cases in Table 5.18, where the multiplicity of FEAS classes obtained here differ from the multiplicity of FAS classes reported in [53]. These cases are boldfaced in the table. We believe that the source of these differences are either typographical error, or some FASs that are not elementary. In addition, it took the exhaustive search of [53] more than 5 hours to generate the fully absorbing sets of each of the two codes on a desktop computer with an Intel Core2 2.4 GHz CPU with 2GB memory [53].

Table 5.19 lists the multiplicity of instances of LETSs in all the nonempty classes within the range $a \leq 12$ and $b \leq 5$, for Codes $\mathcal{C}_9$ and $\mathcal{C}_{10}$. These codes are $(3, 6)$-regular, both with $g = 6$, and with medium and large block lengths: 4000 and $20,000$, respectively. The search algorithm to find instances of LETSs in the range $a \leq 12$ and $b \leq 5$, is based on the information provided in Table 5.6. For each code, all the instances of LETSs in this range are enumerated in less than 9 minutes. From these examples, it can be seen that the dpl-based search algorithm is able to enumerate the instances of dominant LETSs of codes with large block length in a guaranteed fashion, efficiently. Interestingly, despite the large difference between the block lengths of the two codes, the runtimes of the dpl-based search for the two codes are not very different. In comparison, for the dot-based algorithm to search the instances of LETS structures of these codes with $a \leq 12$ and $b \leq 5$, in an exhaustive fashion, the algorithm requires the enumeration of s_{10} and s_{11} structures. This imposes huge computational complexity and memory requirements on the dot-based search algorithm (more than a day of runtime and 8-GB memory). On the other hand, the dpl-search only needs to enumerate prime cycles up to s_6. Moreover, in the dpl-based search, for finding the instances of potentially dominant LETSs, most of the expansion techniques with relatively high complexity (large m) are needed to be applied to the instances of LETS in classes with small b, where such classes happen to be usually empty. For example, in $\mathcal{C}_{10}$, there is no LETS in the $(4, 2)$, $(5, 1)$, $(6, 2)$, $(7, 1)$ and $(8, 2)$ classes.

Table 5.19: Multiplicities of LETS and FEAS structures of Codes $\mathcal{C}_9$ and $\mathcal{C}_{10}$ within the range $a \leq 12$ and $b \leq 5$

	$\mathcal{C}_9$						$\mathcal{C}_{10}$					
(a,b)	Primes				Total	Total	Primes				Total	Total
class	s_3	s_4	s_5	s_6	LETS	FEAS	s_3	s_4	s_5	s_6	LETS	FEAS
(3,3)	171	-	-	-	171	169	161	-	-	-	161	161
(4,2)	1	-	-	-	1	1	-	-	-	-	0	0
(4,4)	-	1219	-	-	1219	1192	-	1260	-	-	1260	1256
(5,3)	-	21	-	-	21	21	-	2	-	-	2	2
(5,5)	-	-	9935	-	9935	9526	-	-	10046	-	10046	9952
(6,4)	278	198	-	-	476	456	49	46	-	-	95	95
(7,3)	5	5	-	-	10	10	0	0	-	-	0	0
(7,5)	3708	2911	1042	-	7661	7359	729	609	202	-	1540	1528
(8,4)	109	74	0	-	183	173	2	2	0	-	4	4
(9,3)	2	2	0	-	4	4	0	0	0	-	0	0
(9,5)	2316	1360	325	-	4001	3847	79	71	17	-	167	165
(10,4)	45	29	0	-	74	71	1	0	0	-	1	1
(11,3)	0	1	0	-	1	1	0	0	0	-	0	0
(11,5)	1371	781	123	3	2278	2189	2	6	0	0	8	8
(12,4)	26	10	0	0	36	36	0	0	0	0	0	0
dpl	7 min.						9 min.					
dot	-						-					

Multiplicities of instances of LETSs and FEAS in all the nonempty classes within the range $a \leq 12$ and $b \leq 4$, for Codes $\mathcal{C}_{11}$, $\mathcal{C}_{12}$ and $\mathcal{C}_{13}$ are listed in Table 9.3. These codes have $d_{\mathrm{v}} = 3$ and $g = 8$. The search algorithm to find all the instances of LETSs in the range of interest is based on the information provided in Table 5.8.

Code $\mathcal{C}_{11}$ is the Tanner $(155, 64)$ code [80] with $d_{\mathrm{c}} = 5$. The error-prone structures of this code have been widely investigated [47], [96], [53], [11], [46]. It is well-known that the LETS structures in the $(8, 2)$ and $(10, 2)$ classes are the dominant LETS structures of this code. It takes the dpl-based search algorithm 442 seconds to find all the instances of LETS structures reported in Table 9.3 for this code. However, it took the dot-based algorithm 45 minutes to find almost all the instances of LETS structures of $\mathcal{C}_{11}$ within this range (prime structures of size 10 are not considered), and more than a day to find the exhaustive list of instances of LETS structures in this table. If we limit the range of search to $a \leq 10$ and $b \leq 4$, it takes the proposed algorithm only 28 seconds to find all the instances of LETS structures of $\mathcal{C}_{11}$ within this range, based on the information provided in Table 5.7. However, it took the dot-based algorithm 392 seconds to find all the instances of LETS structures of $\mathcal{C}_{11}$

Table 5.20: Multiplicities of LETS and FEAS structures of Codes $\mathcal{C}_{11}$, $\mathcal{C}_{12}$ and $\mathcal{C}_{13}$ within the range $a \leq 12$ and $b \leq 4$

	$\mathcal{C}_{11}$						$\mathcal{C}_{12}$						$\mathcal{C}_{13}$					
(a,b)	Primes			Total	Total	[53]	Primes			Total	Total	[53]	Primes			Total	Total	[53]
class	s_4	s_5	s_6	LETS	FEAS	FAS	s_4	s_5	s_6	LETS	FEAS	FAS	s_4	s_5	s_6	LETS	FEAS	FAS
(4,4)	465	-	-	465	0	0	802	-	-	802	760	760	2	-	-	2	2	2
(5,3)	155	-	-	155	155	155	14	-	-	14	14	14	0	-	-	0	0	0
(6,4)	930	-	-	930	0	0	985	-	-	985	849	849	1	-	-	1	1	1
(7,3)	930	-	-	930	0	0	57	-	-	57	47	47	0	-	-	0	0	0
(8,2)	465	-	-	465	465	465	5	-	-	5	4	4	0	-	-	0	0	0
(8,4)	4650	465	-	5115	1395	1395	2414	215	-	2629	2270	2270	2	47	-	49	46	nr
(9,1)	0	-	-	0	0	0	1	-	-	1	1	1	0	-	-	0	0	0
(9,3)	1860	0	-	1860	930	930	152	4	-	156	146	146	0	1	-	1	1	nr
(10,2)	1395	0	-	1395	1395	1395	6	0	-	6	6	6	0	0	-	0	0	0
(10,4)	25575	3720	-	29295	17670	17670	7311	1557	-	8868	7399	nr	0	173	-	173	148	nr
(11,3)	6200	0	-	6200	5270	5270	551	77	-	628	577	nr	0	9	-	9	9	nr
(12,2)	930	0	-	930	930	930	25	2	-	27	26	26	0	0	-	0	0	0
(12,4)	144615	30225	6045	180885	115320	nr†	24857	7062	630	32549	26936	nr	1	461	50	512	466	nr
dpl	442 sec. / 28 sec. *						561 sec. / 38 sec.						157 sec. / 17 sec.					
dot	- / 392 sec.						- / 1563 sec.						- / 1466 sec.					

† "nr" stands for *not reported*.
* Runtime to find all the instances of LETS structures reported in this table/runtime to find instances in the first 10 classes reported in this table.

within the range $a \leq 10$ and $b \leq 4$.

Codes $\mathcal{C}_{12}$ and $\mathcal{C}_{13}$ are $(3,6)$-regular LDPC codes constructed by the PEG algorithm [43]. The dpl-based search algorithm finds all the instances of LETSs for codes $\mathcal{C}_{12}$ and $\mathcal{C}_{13}$ within the range $a \leq 12$ and $b \leq 4$, in 561 and 157 seconds, respectively. Again, the exhaustive search of LETSs for both codes within the range $a \leq 12, b \leq 4$, is out of the reach of the dot-based search algorithm. If we limit the range of search to $a \leq 10$ and $b \leq 4$, it takes the proposed algorithm only 38 and 17 seconds to find all the instances of LETSs of $\mathcal{C}_{12}$ and $\mathcal{C}_{13}$ within this range, respectively. It, however, takes the dot-based algorithm 1563 and 1466 seconds to find all the instances of LETSs of $\mathcal{C}_{12}$ and $\mathcal{C}_{13}$, respectively, within the range $a \leq 10$ and $b \leq 4$.

In Table 9.3, we have also reported the multiplicity of instances of FASs obtained by the exhaustive search algorithm of [53]. It can be seen that the results of [53] match those obtained here. One should, however, note that while the range of classes reported in Table 9.3 is much larger than the range of classes reported in [53], it took the exhaustive search of [53] more than 5 hours, compared to only a few minutes for the proposed algorithm, to generate the fully absorbing sets of each of these codes on

Table 5.21: Multiplicities of LETS and FEAS structures of $\mathcal{C}_{14}$ within the range $a \leq 14$ and $b \leq 6$

	$\mathcal{C}_{14}$				
(a,b)	Primes			Total	Total
class	s_4	s_5	s_6	LETS	FEAS
(4,4)	1320	-	-	1320	1320
(5,5)	-	11088	-	11088	11088
(6,6)	-	-	106920	106920	104280
(7,5)	5280	2640	-	7920	7920
(8,6)	80520	51480	14520	146520	138600
(9,5)	0	2640	-	2640	2640
(10,6)	100320	39600	7920	147840	129360
(11,5)	5280	0	0	5280	0
(12,4)	1320	0	0	1320	1320
(12,6)	73040	52800	9240	135080	128480
(13,5)	2640	0	0	2640	0
(14,4)	1320	0	0	1320	1320
dpl	42 min.				
dot	-				

a desktop computer with an Intel Core2 2.4 GHz CPU with 2GB memory [53].

Code $\mathcal{C}_{14}$ is the Margulis $(2640, 1320)$ code [59], with $g = 8$. It is well-known that the most dominant LETS structures of this code are in (12,4) and (14,4) LETS classes. The proposed search algorithm finds all the instances of LETSs in Table 5.21 for this code in about 42 minutes. It is worth mentioning that [53] only reports the multiplicity of FAS structures in the $(4,4), (5,5)$ classes of $\mathcal{C}_{14}$. The results of [53] match the results reported in the first two rows of Table 5.21.

Table 5.22 lists the multiplicity of instances of LETSs, EASs and FEASs in all the nonempty classes within the range $a \leq 6$ and $b \leq 4$, for Codes $\mathcal{C}_{15}$-$\mathcal{C}_{19}$. These are five high-rate array-based codes with $g = 6$, $d_{\text{v}} = 4$, and $d_{\text{c}} = 7, 13, 17, 19, 23$, respectively. It was shown in [17] that array-based codes with $d_{\text{v}} = 4$ and $d_{\text{c}} > 7$, do not contain any absorbing sets (ASs) in the $(4, 4)$ class. This is matched with our results for $(4, 4)$ EASs. Moreover, from Table 5.22, one can see that the array-based code with $d_{\text{v}} = 4$ and $d_{\text{c}} = 7$, has 294 EASs, but no FEAS in the $(4, 4)$ class. Lemmas 6 and 7 in [17] indicate that for the array-based codes with $d_{\text{v}} = 4$ and $d_{\text{c}} > 19$, there is no $(5, b)$ and $(6, 2)$ absorbing sets. Based on Table V of [46], for a variable-regular code with $d_{\text{v}} = 4$ and $g = 6$, the $(5, 0)$, $(5, 2)$ and $(5, 4)$ classes are the only classes with size 5 that can

Table 5.22: Multiplicities of LETS, EAS and FEAS structures of Codes $\mathcal{C}_{15}$, $\mathcal{C}_{16}$, $\mathcal{C}_{17}$, $\mathcal{C}_{18}$, $\mathcal{C}_{19}$ within the range $a \leq 6$ and $b \leq 4$

	$\mathcal{C}_{15}$			$\mathcal{C}_{16}$			$\mathcal{C}_{17}$			$\mathcal{C}_{18}$			$\mathcal{C}_{19}$		
(a,b) class	Total LETS	Total EAS	Total FEAS	Total LETS	Total EAS	Total FEAS	Total LETS	Total EAS	Total FEAS	Total LETS	Total EAS	Total FEAS	Total LETS	Total EAS	Total FEAS
(4,4)	294	294	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,4)	1176	0	0	0	0	0	0	0	0	12996	12996	0	0	0	0
(6,2)	588	588	588	0	0	0	0	0	0	0	0	0	0	0	0
(6,4)	4116	1764	0	30420	30420	0	23120	23120	18496	110466	32490	25992	58190	58190	46552
dpl	1 min.			2 min.			4 min.			7 min.			11 min.		
dot	1 min.			2 min.			4 min.			7 min.			11 min.		

have elementary absorbing sets (EASs). The results of Table 5.22 thus reveal that $\mathcal{C}_{18}$ with $d_c = 19$ is the only array-based code with EASs of size 5, and that the only class of size 5 for which this code has some EASs is the $(5, 4)$ class. Also, our results in Table 5.22 complement those of [17], and demonstrate that the array-based codes with $d_v = 4$ and $d_c > 7$ do not contain any EASs in the $(6, 2)$ class. Since the *dot* expansion is the only expansion technique used in the dpl-based search algorithm for the array-based codes, this algorithm is the same as the dot-based search algorithm for these codes.

Table 5.23 lists the multiplicity of instances of LETSs, EASs and FEASs in all the nonempty classes within the range $a \leq 8$ and $b \leq 6$, for Codes $\mathcal{C}_{20}$ and $\mathcal{C}_{21}$. These are two high-rate codes with $d_v = 4$, $g = 6$ and block lengths $n = 4096$ and $n = 16,383$, respectively. The search algorithm to find all the instances of LETSs in the range $a \leq 8$ and $b \leq 6$, is based on the information provided in Table 5.10. All the instances of LETS structures in this range for $\mathcal{C}_{20}$ and $\mathcal{C}_{21}$ are enumerated in about 68 and 133 minutes, respectively. For the dot-based search algorithm to find all the instances of LETSs in the range of interest in a guaranteed fashion, it requires to enumerate s_6 structures. This enumeration alone imposes huge computational complexity and memory requirements to the dot-based search algorithm (more than a day of runtime and 8-GB of memory).

We perform Monte Carlo simulations on code $\mathcal{C}_{20}$ with a 5-bit quantized min-sum decoder (with clipping threshold 7.5) over AWGN channel to obtain 100 block errors at different signal-to-noise ratios (SNR). The LETSs reported in Table 5.23 were used to estimate the error floor of the code using the importance sampling technique described in [8]. Fig. 5.6 shows the Monte Carlo simulation results for the frame

Table 5.23: Multiplicities of LETS, EAS and FEAS structures of Codes $\mathcal{C}_{20}$ and $\mathcal{C}_{21}$ within the range $a \leq 8$ and $b \leq 6$

	$\mathcal{C}_{20}$					$\mathcal{C}_{21}$				
(a,b)	Primes		Total	Total	Total	Primes		Total	Total	Total
class	s_3	s_4	LETS	EAS	FEAS	s_3	s_4	LETS	EAS	FEAS
(3,6)	43531	-	43531	0	0	119233	-	119233	0	0
(4,4)	15	-	15	15	13	10	-	10	10	6
(4,6)	21383	-	21383	0	0	29001	-	29001	0	0
(5,4)	9	-	9	7	4	5	-	5	1	1
(5,6)	14622	-	14622	0	0	9796	-	9796	0	0
(6,4)	10	-	10	7	5	2	-	2	2	0
(6,6)	11553	1403	12956	913	246	3815	530	4345	307	161
(7,4)	10	0	10	6	5	2	0	2	0	0
(7,6)	10941	1837	12778	1779	476	1859	337	2196	288	167
(8,4)	12	1	13	11	6	1	0	1	0	0
(8,6)	11479	2659	14138	2612	741	970	188	1158	191	102
dpl	68 min.					133 min.				
dot	-					-				

error rate (FER) and the corresponding error floor estimation based on importance sampling. As can be seen in Fig. 5.6, the estimation closely matches the Monte Carlo simulation, verifying the dominance of the 15 LETSs in the $(4, 4)$ class. Except the LETSs in the $(4, 4)$ and $(5, 4)$ classes, the contributions of all the other LETSs listed in Table 5.23 are negligible. Therefore, all the dominant LETSs of this code are in the range of $a \leq 6$ and $b \leq 4$. If we limit the range of interest to $a \leq 6$ and $b \leq 4$, it takes the proposed search algorithm less than 45 seconds to find all the LETSs of $\mathcal{C}_{20}$ within this range.

Table 5.24 lists the multiplicity of instances of LETSs, EASs and FEASs in all the nonempty classes within the range $a \leq 10$ and $b \leq 10$, for $\mathcal{C}_{22}$ and $\mathcal{C}_{23}$. These are $(4, 8)$-regular LDPC codes, with $g = 6$, and with block lengths of $n = 4000$ and $n = 8000$, respectively. The search algorithm to find all the instances of LETSs in this range is based on the information provided in Table 5.11. It can be observed that $\mathcal{C}_{22}$ and $\mathcal{C}_{23}$ have one and no instance of EAS in this range, respectively. The dominant LETSs of $\mathcal{C}_{23}$, which are included in those reported in Table 5.24, are used in [77] to accurately estimate the error floor under quantized iterative decoders. The accurate estimation reveals that error-prone structures of this code are not absorbing sets. If we limit the range of interest to $a \leq 9$ and $b \leq 8$, it takes the proposed search

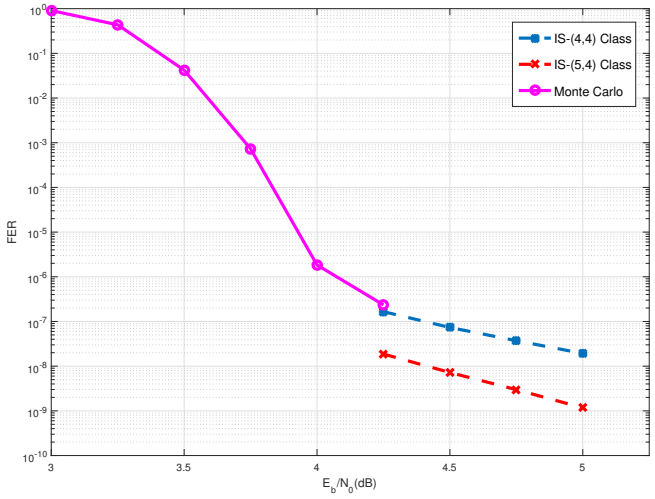


Figure 5.6: Simulation results of $\mathcal{C}_{20}$, decoded by a 5-bit min-sum decoder and the corresponding error floor estimation.

Table 5.24: Multiplicities of LETS, EAS and FEAS structures of Codes $\mathcal{C}_{22}$ and $\mathcal{C}_{23}$ within the range $a \leq 10$ and $b \leq 10$

	$\mathcal{C}_{22}$						$\mathcal{C}_{23}$					
(a,b)	Primes			Total	Total	Total	Primes			Total	Total	Total
class	s_3	s_4	s_5	LETS	EAS	FEAS	s_3	s_4	s_5	LETS	EAS	FEAS
(3,6)	1563	-	-	1563	0	0	1620	-	-	1620	0	0
(4,6)	91	-	-	91	0	0	55	-	-	55	0	0
(4,8)	-	24269	-	24269	0	0	-	24107	-	24107	0	0
(5,6)	2	-	-	2	0	0	2	-	-	2	0	0
(5,8)	4080	560	-	4640	0	0	2011	292	-	2303	0	0
(5,10)	-	-	402513	402513	0	0	-	-	406289	406289	0	0
(6,8)	495	28	65	588	0	0	164	5	27	196	0	0
(6,10)	111540	74004	-	185544	0	0	50073	37172	-	96525	0	0
(7,8)	51	2	0	53	0	0	10	0	0	10	0	0
(7,10)	33205	6589	-	39794	0	0	9009	1689	-	10698	0	0
(8,8)	5	0	-	5	0	0	0	0	-	0	0	0
(8,10)	6248	754	4	7006	0	0	1030	84	2	1116	0	0
(9,8)	1	0	-	1	0	0	0	0	-	0	0	0
(9,10)	1038	107	0	1145	0	0	96	5	0	101	0	0
(10,10)	170	15	0	185	1	1	13	0	0	13	0	0
dpl	63 min.						51 min.					
dot	-						-					

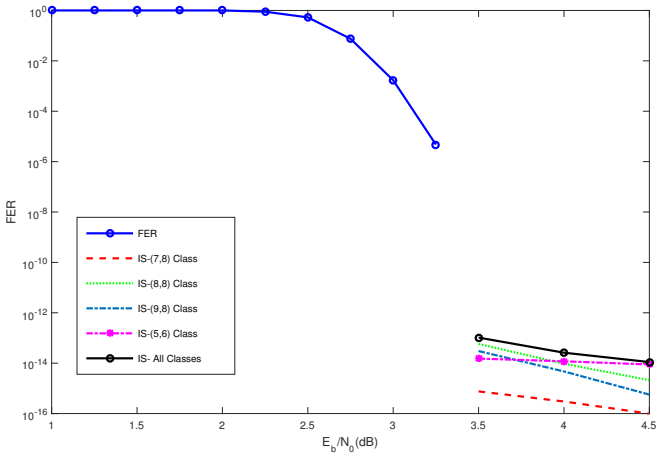


Figure 5.7: Simulation results of $\mathcal{C}_{22}$, decoded by the 5-bit min-sum decoder and the corresponding error floor estimation.

algorithm only 4 minutes to find all the LETSs of $\mathcal{C}_{22}$ within this range. It is worth mentioning that to find all the instances of LETSs of $\mathcal{C}_{22}$ within the range $a \leq 9$ and $b \leq 8$, by the dot-based search algorithm, all the instances of s_6, s_7 and s_8, need to be enumerated. It takes more than 7 hours to only enumerate instances of s_7 in the Tanner graph of this code, and enumerating instances of s_8 is not even feasible on the desktop computer.

We perform Monte Carlo simulations on code $\mathcal{C}_{22}$ with a 5-bit quantized min-sum decoder (with clipping threshold 7.5) over AWGN channel to obtain 100 block errors at different SNRs. The LETSs reported in Table 5.24 were used to estimate the error floor of the code using the importance sampling technique. Fig. 5.6 shows the Monte Carlo simulation results for FER and the corresponding error floor estimation based on importance sampling. It takes the Monte Carlo simulation more than two weeks to obtain 100 block errors at SNR 3.25 (the error floor region of this code is out of reach of Monte Carlo simulation on any desktop computer). Also, as can be seen in Fig. 5.7, the dominant structures are LETSs in the $(5, 6)$, $(8, 8)$ and $(9, 8)$ classes. To the best of our knowledge, there exist no exhaustive search algorithm in the literature to find the dominant trapping sets of large block length LDPC codes with $d_{\text{v}} = 4$ such as $\mathcal{C}_{22}$. However, it takes the *dpl*-based search algorithm only 4 minutes to find all the LETSs of $\mathcal{C}_{22}$ in the range of $a \leq 9$ and $b \leq 8$.

Table 5.25 lists the multiplicity of instances of LETSs, EASs and FEASs in all the

Table 5.25: Multiplicities of LETS, EAS and FEAS Structures of Code $\mathcal{C}_{24}$ within the range $a \leq 9$ and $b \leq 11$

	$\mathcal{C}_{24}$				
(a,b)	Primes		Total	Total	Total
class	s_3	s_4	LETS	EAS	FEAS
(3,9)	7876	-	7876	0	0
(4,8)	30	-	30	30	30
(4,10)	8446	-	8446	0	0
(5,9)	98	-	98	36	34
(5,11)	14873	-	14873	0	0
(6,8)	1	-	1	1	1
(6,10)	321	-	321	118	116
(7,9)	5	-	5	1	1
(7,11)	1132	9	1141	371	337
(8,8)	1	0	1	1	1
(8,10)	31	0	31	20	18
(9,9)	4	0	4	1	1
(9,11)	165	2	167	76	68
dpl	6 min.				
dot	-				

nonempty classes within the range $a \leq 9$ and $b \leq 11$, for $\mathcal{C}_{24}$. This is a $(5, 10)$-regular LDPC code with $g = 6$, and block length $n = 816$. All the instances of LETSs of this code in the range $a \leq 9$ and $b \leq 11$, are enumerated in about 6 minutes. It can be seen that for this code, as a result of relatively large d_{v}, no LETS with $b < a$ exists in the range of interest. Moreover, there is no FEAS for this code in the range $b < a$. It is reported in [77] that using dominant LETSs of $\mathcal{C}_{24}$, included in Table 5.25, one can obtain a more accurate estimation of the error floor in comparison to using only EASs.

To the best of our knowledge, there are only two exhaustive search algorithms in the literature for finding the error-prone structures of variable-regular LDPC codes which cover a relatively wide range of trapping set classes, rates, block lengths and variable degrees: the algorithm of [53], and that of [46]. The work in [53] is limited to only fully absorbing sets (FASs). Also, the regular LDPC codes which were studied in [53] are all codes with $d_{\text{v}} = 3$, short block lengths ($n < 2000$) and rates equal to or less than 0.5. The variable-regular LDPC codes studied in [53] are Codes $\mathcal{C}_7$, $\mathcal{C}_8$, $\mathcal{C}_{11}$, $\mathcal{C}_{12}$, $\mathcal{C}_{13}$ and $\mathcal{C}_{14}$ in this chapter. While, it takes the dpl-based search algorithm less than 42 minutes to find all the dominant LETSs and FEASs of each of these

codes, the runtime to find FASs of these codes in [53] (in smaller ranges of a and b values than those considered in this work) ranges from 5 to 48 hours using a desktop computer with an Intel core2 2.4-GHz CPU with 2-GB memory. Compared to the LSS- or dot-based algorithm of [46], [28], as we discussed in the tables, the runtime and memory requirements of dpl-based search are up to three orders of magnitude smaller.

To the best of our knowledge, the most versatile algorithm for finding LETSs of LDPC codes is that of [47]. This algorithm, however, is not exhaustive in the sense that it does not provide any guarantee that all the instances of (a, b) LETS structures with a and b values within a given range of interest are found. For example, Codes $\mathcal{C}_{11}$, $\mathcal{C}_{12}$, and $\mathcal{C}_{14}$ have also been studied in [47], and their LETSs (EASs) are presented in Tables II, I and III of [47], respectively. The comparison of the multiplicity of LETSs in different classes reported in this work, and that reported in [47], for each of these codes, reveals that for each code, there are some classes for which the multiplicity reported in [47] is smaller than that of this work. This means that for such classes, the algorithm of [47] has not been able to find all the LETSs in that class. These classes for Codes $\mathcal{C}_{11}$, $\mathcal{C}_{12}$, and $\mathcal{C}_{14}$ are $\{(9,3),(11,3)\}$, $\{(8,4),(10,4),(11,3),(12,2),(12,4)\}$, and $\{(8,6),(10,6)\}$, respectively.

Chapter 6

Characterization/Search of ETSs of Irregular LDPC Codes

6.1 Introduction

In this chapter, we study the graphical structure of the ETSs in irregular LDPC codes. There are very few works on the trapping sets of irregular LDPC codes [1], [47], [53], [23]. This is despite the fact that these codes are popular in many applications due to their superior performance over the regular codes in the waterfall region [67]. In fact, irregular LDPC codes have been already adopted in a number of standards [101], [100], [105], [104], [102]. One main reason for the lack of results on trapping sets of irregular codes is the variety of variable node degrees that makes the identification and characterization of trapping sets of different classes a seemingly impossible task [47]. In the following, we review the main existing literature related to trapping sets of irregular codes: In [1], using the modified impulse algorithm, the authors devised a technique to find a non-exhaustive list of trapping sets of a given irregular LDPC code. In [47], by examining the relationships between cycles and trapping sets, Karimi and Banihashemi proposed an efficient search algorithm to find the dominant ETSs of a given irregular LDPC code. Although the proposed algorithm in [47] can find ETSs with sufficiently large size, it provides no guarantee that the obtained list of ETSs is exhaustive. Using the branch-&-bound principle, Kyung and Wang [53] proposed an exhaustive search algorithm to enumerate the fully absorbing sets (FASs) of short irregular LDPC codes. The proposed algorithm, however, becomes quickly infeasible to use as the block length, n, and the size of the FASs, a, are increased. In general, the reach of the algorithm is limited to $n < 1000$ and $a < 7$, or $n < 1000$ and $a < 14$,

if the number of unsatisfied check nodes is limited to $b < 3$. Recently, Falsafain and Mousavi [23], proposed a branch-&-bound algorithm to find the ETSs of irregular LDPC codes. The presented 0-1 integer linear programming (ILP) formulation in [23] provided a tighter linear programming relaxation in comparison with the one in [53]. However still the exhaustive search algorithm in [23] is only applicable to short-to-moderate length LDPC codes. To the best of our knowledge, the algorithms proposed in [53] and [23] are the only exhaustive search algorithms of error-prone structures in irregular codes. Also, to the best of our knowledge, almost all the structures reported in the literature as error-prone structures of irregular LDPC codes are ETSs, with a large majority being LETS structures.

In this chapter, first, we focus on LETS structures of irregular codes in Sections 6.2 and 6.3. We then study ETS structures which are not leafless in Section 6.4. Finally, numerical results are presented in Section 6.5.

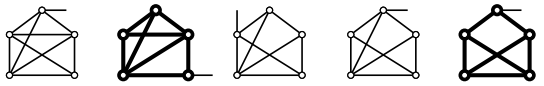
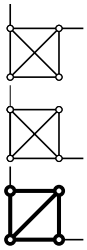
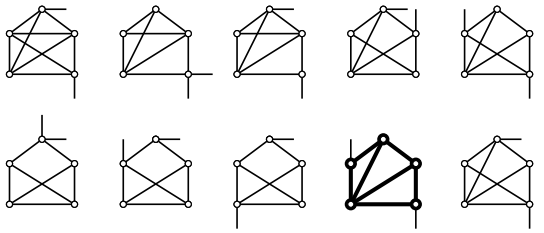
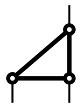
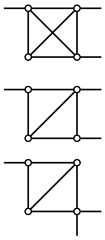
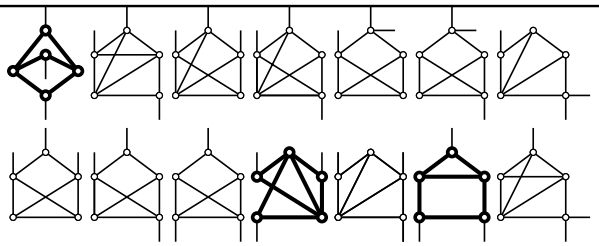
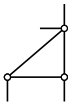
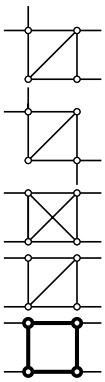
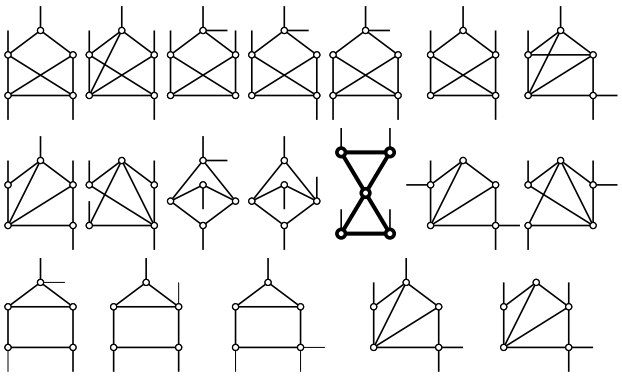
6.2 Dpl-Based Characterization/Search of LETS Structures in Irregular LDPC Codes

In irregular Tanner graphs, for a given class of LETSs, the variety of non-isomorphic LETS structures would increase significantly compared to that of variable-regular Tanner graphs. This is due to the variety of the degrees of variable nodes involved in LETS structures.

Example 14. *Suppose that one is interested in characterizing the non-isomorphic* (a, b) *LETS structures of irregular Tanner graphs with variable degrees 3 and 4 in the interest range of* $a \leq 5$ *and* $b \leq 4$. *Table 6.1 shows the quasi-normal graph representation of all the possible non-isomorphic LETS structures in this range. Comparing the information of this table with that of Tables 5.3 and 5.9, for variable-regular graphs with* $d_v = 3$ *and* $d_v = 4$, *respectively, shows the considerable difference in the number of non-isomorphic LETS structures in this range for variable-regular versus irregular graphs. As an example, Table 6.1 shows that there are 19 non-isomorphic LETS structures in the* $(5, 4)$ *class. This number for the* $(5, 4)$ *class in Table 5.3 is only 2.*

It can be seen that by increasing the range of a and b values or by having a larger variety of variable node degrees, the number of non-isomorphic structures increases

Table 6.1: Non-isomorphic (a, b) LETS structures of an irregular Tanner graph with variable degrees 3 and 4, in the range of $a \leq 5$ and $b \leq 4$

	$a = 3$	$a = 4$	$a = 5$
$b = 0$			
$b = 1$			
$b = 2$			
$b = 3$			
$b = 4$			

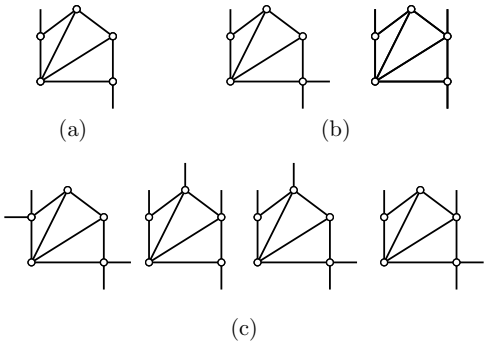


Figure 6.1: Seven non-isomorphic LETS structures in the (a) $(5, 2)$, (b) $(5, 3)$ and (c) $(5, 4)$ classes, all with the same normal graph .

rapidly. An important observation, however, is that despite the large number of non-isomorphic quasi-normal graphs in each class, they are all projected to only a few normal graphs. In Table 6.1, the non-isomorphic normal graphs, which are the projections of all the quasi-normal graphs in the table, are boldfaced. For example, the boldfaced graph in the $(5, 2)$ class is the projection of 7 LETS structures, where one, two and four structures are in the $(5, 2)$, $(5, 3)$ and $(5, 4)$ classes, respectively. These 7 LETS structures are presented in Fig. 6.1.

6.2.1 Dpl characterization of LETS structures in irregular graphs

In the following, we demonstrate, through a sequence of intermediate results, that the dpl characterization of (a, b) LETS structures of variable-regular Tanner graphs of a properly selected variable degree d_{v}, over a properly chosen range $a \leq a'_{max}$ and $b \leq b'_{max}$ can be used to exhaustively cover all the normal graphs of all the non-isomorphic (a, b) LETS structures of irregular Tanner graphs with a given degree distribution $\lambda(x)$ over any desired range of $a \leq a_{max}$ and $b \leq b_{max}$. (The values a'_{max}, b'_{max}, and d_{v} are functions of a_{max}, b_{max} and $\lambda(x)$.) This means that a dpl characterization can also be established for irregular graphs.

Suppose that $\mathcal{L}^a_{d_{\mathrm{v}}}$ is the set of all non-isomorphic LETS structures of size a in the Tanner graph of a variable-regular graph with variable degree d_{v}. It is easy to see that the structures in $\mathcal{L}^a_{d_{\mathrm{v}}}$ can have b values in the range $0 \leq b \leq a(d_{\mathrm{v}} - 2)$. The following proposition establishes a relationship between the sets $\mathcal{L}^a_{d_{\mathrm{v}}}$, for different values of d_{v}.

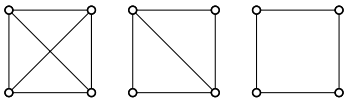


Figure 6.2: All the non-isomorphic LETS structures with size $a = 4$ in a variable-regular Tanner graph with $d_{\mathrm{v}} \geq 3$.

Proposition 6. *In the space of normal graphs, if* $d_{\mathrm{v}} < a - 1$*, then* $\mathcal{L}^a_{d_{\mathrm{v}}} \subset \mathcal{L}^a_{a-1}$*, and if* $d_{\mathrm{v}} > a - 1$*, then* $\mathcal{L}^a_{d_{\mathrm{v}}} = \mathcal{L}^a_{a-1}$.

Proof. For the first part, consider an arbitrary element $\mathcal{S}$ of $\mathcal{L}^a_{d_{\mathrm{v}}}$. The LETS $\mathcal{S}$ has a nodes, each node having a degree d in the range $2 \leq d \leq d_{\mathrm{v}}$. Since $\mathcal{L}^a_{a-1}$ includes *all* the LETS structures with a nodes, where each node can have a degree in the range $[2, a-1]$, and since $d_{\mathrm{v}} < a - 1$, the structure $\mathcal{S}$ is also in $\mathcal{L}^a_{a-1}$, and thus $\mathcal{L}^a_{d_{\mathrm{v}}} \subset \mathcal{L}^a_{a-1}$.

For the second part, following similar steps as in the proof of the first part, it can be shown that if $a - 1 < d_{\mathrm{v}}$, then $\mathcal{L}^a_{a-1} \subset \mathcal{L}^a_{d_{\mathrm{v}}}$. Now, consider a structure $\mathcal{S}$ in $\mathcal{L}^a_{d_{\mathrm{v}}}$ that is not in $\mathcal{L}^a_{a-1}$. Structure $\mathcal{S}$ must then have at least one node v with degree strictly larger than $a - 1$. This is, however, impossible as node v must be connected to at least a other nodes, while there are only $a - 1$ other nodes in $\mathcal{S}$. We thus have $\mathcal{L}^a_{d_{\mathrm{v}}} = \mathcal{L}^a_{a-1}$.

■

It is important to note that even if two sets of LETS structures with different variable degrees are identical (in the space of normal graphs), they still correspond to different sets of classes. This is explained in the following examples.

Example 15. *Fig. 6.2 shows all the non-isomorphic LETS structures with size* $a = 4$ *in a variable-regular Tanner graph with* $d_{\mathrm{v}} = 3$ *and* $g = 6$*. Based on the second part of Proposition 6, these structures are also all the non-isomorphic LETS structures with size* $a = 4$ *in a variable-regular Tanner graph with* $d_{\mathrm{v}} > 3$ *and* $g = 6$*. For variable-regular graphs with* $d_v = 3$ *and* $g = 6$*, the structures in Figs. 6.2 (a), (b) and (c) are the only structures in the* $(4, 0)$*,* $(4, 2)$ *and* $(4, 4)$ *classes, respectively (see Table 5.3). The same structures, for variable-regular graphs with* $d_v = 4$ *and* $g = 6$*, are the only structures in the* $(4, 4)$*,* $(4, 6)$ *and* $(4, 8)$ *classes, respectively (see Table 5.9).*

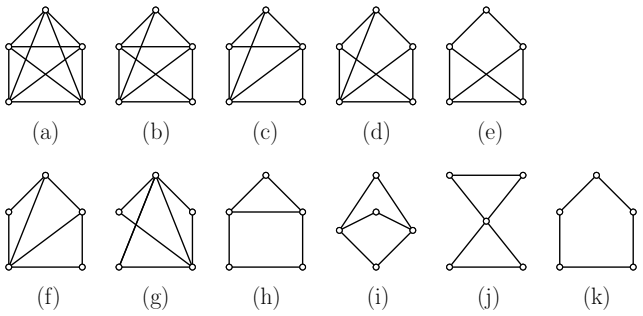


Figure 6.3: All the non-isomorphic LETS structures with size $a = 5$ in a variable-regular Tanner graph with $d_{\mathrm{v}} \geq 4$.

Table 6.2: Classes of non-isomorphic LETS structures in Fig. 6.3 for variable-regular graphs with $d_{\mathrm{v}} = 3, 4$ and 5

Fig. 6.3	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)
$d_{\mathrm{v}} = 3$	-	-	-	-	(5,1)	-	-	(5,3)	(5,3)	-	(5,5)
$d_{\mathrm{v}} = 4$	(5,0)	(5,2)	(5,4)	(5,4)	(5,6)	(5,6)	(5,6)	(5,8)	(5,8)	(5,8)	(5,10)
$d_{\mathrm{v}} = 5$	(5,5)	(5,7)	(5,9)	(5,9)	(5,11)	(5,11)	(5,11)	(5,13)	(5,13)	(5,13)	(5,15)

Example 16. *Fig. 6.3 provides all the non-isomorphic LETS structures with size $a = 5$ in a variable-regular Tanner graph with $d_{\mathrm{v}} = 4$ and $g = 6$. Based on the second part of Proposition 6, these structures are also all the non-isomorphic LETS structures with size $a = 5$ in a variable-regular Tanner graph with $d_{\mathrm{v}} > 4$ and $g = 6$. For example, $\mathcal{L}_5^5 = \mathcal{L}_4^5$. Moreover, based on the first part of Proposition 6, $\mathcal{L}_3^5 \subset \mathcal{L}_4^5$. Table 6.2 shows the classes of these structures in variable-regular Tanner graphs with $d_{\mathrm{v}} = 3, 4, 5$, and $g = 6$.*

The following proposition explains how different LETS structures of an irregular Tanner graph, corresponding to different quasi-normal graphs, are mapped (projected) to normal graphs of LETS structures of a variable-regular Tanner graph.

Proposition 7. *Any (quasi-normal graph of a) LETS structure $\mathcal{S}$ in the (a, b) class of an irregular Tanner graph with variable degree distribution $\lambda(x)$ is mapped (via a surjective mapping) to a (normal graph of a) LETS structure in the (a, b') class of variable-regular Tanner graphs with variable degree $d_{\mathrm{v}} = f$, where f is the largest variable degree in $\lambda(x)$ strictly less than a and $b' = a \times f + b - \sum_{i=1}^{a} d_{v_i}$, with $v_i, i = 1, \ldots, a$, being the variable nodes of $\mathcal{S}$.*

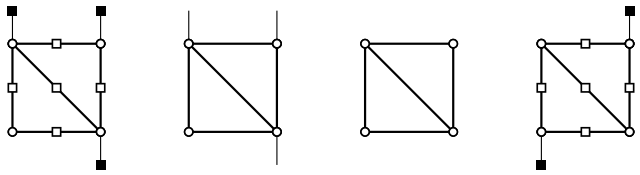


Figure 6.4: (a) Tanner graph, and (b) quasi-normal representations of a LETS structure in the $(4, 3)$ class of an irregular graph with variable degrees $2, 3, 4$; (c) normal graph, and (d) Tanner graph representations of a LETS structure in the $(4, 2)$ class of variable-regular graphs with $d_{\mathrm{v}} = 3$.

Proof. The normal graph representation of $\mathcal{S}$ has a nodes, where each of them is connected to at most f other nodes in $\mathcal{S}$ (f is the largest variable degree in $\lambda(x)$ strictly less than a). Based on Proposition 6, this normal graph is thus a LETS in variable-regular graphs with any variable degree $d_{\mathrm{v}} \geq f$. Selecting the minimum variable degree in this range, i.e., $d_{\mathrm{v}} = f$, we can easily find the class of this structure using Lemma 2. For this, we note that $\mathcal{S}$ has $e = (\sum_{i=1}^{a} d_{v_i} - b)/2$ edges. Based on Lemma 2, the number of unsatisfied check nodes of $\mathcal{S}$ in a variable-regular graph with $d_{\mathrm{v}} = f$ is $b' = a \times f - 2e = a \times f + b - \sum_{i=1}^{a} d_{v_i}$. ■

Example 17. *Fig. 6.4 shows a LETS structure in the $(4, 3)$ class of an irregular graph with variable degrees $2, 3, 4$, and the process of surjective mapping to a LETS structure in the $(4, 2)$ class of a variable-regular graph with $d_{\mathrm{v}} = 3$.*

Proposition 7 describes how the LETS structures of irregular graphs are mapped to LETS structures of variable-regular graphs via the normal graph representation. The following theorem explains how the dpl characterization of LETS structures in variable-regular graphs can be used to characterize the LETS structures of irregular graphs (in a given range of a and b values, exhaustively).

Theorem 3. *The dpl characterization of non-isomorphic LETS structures for variable-regular graphs with $d_{\mathrm{v}} = t$ in (a, b) classes with $a \leq a_{max}$ and $b \leq b_{max} + a_{max}(t - d_{v_{min}})$ is sufficient to generate the normal graphs of all the non-isomorphic (a, b) LETS structures with $a \leq a_{max}$ and $b \leq b_{max}$ of irregular Tanner graphs with variable degree distribution $\lambda(x) = \sum_{i=d_{v_{min}}}^{d_{v_{max}}} \lambda_i x^{i-1}$, where t is the largest variable degree in $\lambda(x)$ strictly less than a_{max}.*

Proof. Proposition 7 describes the mapping between the LETS structures of irregular graphs to those of variable-regular graphs. Based on this proposition, to cover the projections of all the LETS structures with $a \leq a_{max}$ of irregular graphs, the variable node degree of the variable-regular graph needs to be $d_{\mathrm{v}} = t$, where t is the largest variable degree in $\lambda(x)$ strictly less than a_{max}. Moreover, to cover all the LETS classes of the irregular graphs in the interest range of $a \leq a_{max}$ and $b \leq b_{max}$, based on Proposition 7, the maximum value of b' is $b_{max} + a_{max}(t - d_{v_{min}})$. This is obtained by noting that, in Proposition 7, $f = t$, and that b' is maximized by setting a and b to their maximum values a_{max} and b_{max}, respectively, and by minimizing $\sum_{i=1}^{a_{max}} d_{v_i}$ through assuming that all the variable nodes in the LETS have the minimum degree $d_{v_{min}}$.

■

Example 18. *Based on Theorem 3, to cover all the (a, b) LETS structures in the range of $a \leq 7$ and $b \leq 3$, of an irregular graph with variable degrees 3, 4 and 7, one should characterize all the LETS structures in a variable-regular graph with $d_{\mathrm{v}} = 4$ in the range of $a \leq 7$ and $b \leq 10$. This characterization is summarized in Table 6.3. This representation of characterization is similar to that of Chapter 5. Note that to cover this rather small range of a and b values for the irregular graph, one needs to cover a wide range of b values for the corresponding variable-regular graph.*

6.2.2 Dpl-based search algorithm for irregular graphs

In Chapter 5, for variable-regular graphs, the dpl characterization of LETS structures was used as a road-map for the dpl-based search algorithm to find all the instances of LETS structures in any interest range of a and b values in a guaranteed fashion. The search algorithm of Chapter 5 starts by the enumeration of simple cycles. These are the cycles that are identified in the characterization table as the required prime structures. After the enumeration of all the instances of a simple cycle, these instances are expanded recursively to find instances of other LETS structures up to size a_{max}. In each step, after finding a new LETS, the indices of its variable nodes should be saved for subsequent expansions in the next step. The expansion techniques identified by the characterization table should be applied to all the instances of LETS structures in the corresponding classes.

Table 6.3: Characterization of Non-Isomorphic LETS Structures of (a, b) Classes for Variable-regular Graphs with $d_{\mathrm{v}} = 4$ and $g = 6$ for $a \leq a_{max} = 7$ and $b \leq b_{max} = 10$

	$a = 3$	$a = 4$	$a = 5$	$a = 6$	$a = 7$
b = 0	-	-	$s_3(1)$ – – – –	$s_3(1)$ – – – –	$s_3(2)$ – – – –
b = 1	-	-	-	-	-
b = 2	-	-	$s_3(1)$ – – – *dot*	$s_3(3)$ – – – *dot*	$s_3(9)$ – – – –
b = 3	-	-	-	-	-
b = 4	-	$s_3(1)$ – – – *dot*	$s_3(2)$ – – – *dot*	$s_3(7)$ – – – *dot*	$s_3(27), s_4(1)$ – – – –
b = 5	-	-	-	-	-
b = 6	$\boldsymbol{s_3(1)}$ – – – dot, pa_2 pa_3, lo_3^3	$s_3(1)$ – – – dot, pa_2 pa_3	$s_3(3)$ – – – *dot*	$s_3(10), s_4(1)$ – – – *dot*	$s_3(43), s_4(1)$ – – – –
b = 7	-	-	-	-	-
b = 8	-	$\boldsymbol{s_4(1)}$ – – – dot, pa_2	$s_3(2), s_4(1)$ – – – dot, pa_2	$s_3(8), s_4(2)$ – – – *dot*	$s_3(41), s_4(3)$ – – – –
b = 9	-	-	-	-	-
b = 10	-	-	– – – –	$s_3(3), s_4(2)$ – – – *dot*	$s_3(21), s_4(6)$ – – – –

In Subsection 6.2.1, it was shown that the characterization table for variable-regular graphs with variable degree $d_v = t$ in the range of $a \leq a_{max}$ and $b \leq b_{max} + a_{max}(t - d_{v_{min}})$ can be used to characterize the (a, b) LETS structures of irregular graphs with degree distribution $\lambda(x) = \sum_{i=d_{v_{min}}}^{d_{v_{max}}} \lambda_i x^{i-1}$ in the range of $a \leq a_{max}$ and $b \leq b_{max}$, where t is the largest variable degree less than a_{max} in $\lambda(x)$. Unlike the case for variable-regular graphs, in irregular Tanner graphs, however, there is no one-to-one correspondence between a LETS structure and its normal graph, i.e., a normal graph can be the projection of multiple LETS structures in different classes. In the search algorithm for irregular graphs, therefore, for each LETS, in addition to the index of its variable nodes, we need to also keep track of the class of its normal graph in variable-regular graphs with $d_v = t$. Otherwise, the search follows the exact similar steps as in the dpl-based search of Chapter 5. More details are provided in the following.

The dpl-based search algorithm of irregular graphs starts by the enumeration of simple cycles (as identified in the characterization table for the corresponding variable-regular graph with $d_v = t$). For different instances of a simple cycle of a given length, based on the degrees of the variable nodes in the cycles, the cycles would belong to different classes, all with the same a but different b values. In addition to saving the information of each instance (indices of its variable nodes), the class of its normal graph in the characterization table for variable-regular graphs with $d_v = t$ should be saved as well. After the enumeration of all instances of a simple cycle, these instances are expanded recursively to find instances of other LETS structures up to size a_{max} following the dpl-based search algorithm (Algorithm 2). It is important to note that, in the search process, no matter what the actual value of b of an instance of a LETS is, as long as its normal graph is in the (a, b) class of variable-regular graphs with $d_v = t$, the characterization table for the variable-regular graph shows which expansions should be applied to that instance. Also, no matter what the class of the resultant instance is, one only needs to keep track of the class of its normal graph (in the characterization table for variable-regular graphs with $d_v = t$).

Example 19. *Suppose that in an irregular graph with variable degrees* $3, 4$ *and* 7*, there are a set of instances of LETSs in different classes with* $a = 4$ *and* $b = 2, 3, 4, 6, 7$ *and* 8*. Also suppose that all these instances are projected to the normal graph in the* $(4, 4)$ *class of Table 6.3. Applying* dot_2 *to all these instances may result in new*

instances with $a = 5$ and, for example, $b = 1, 2, 5, 7$, and 8. However, based on Proposition 1, the normal graph of all the new instances will be in the $(5, 4)$ class of Table 6.3. Therefore, no matter what the classes of new instances are, the class $(5, 4)$ should be saved for each of them for the next step. This class is the one that is used to determine the expansions that will have to be applied to these instances in the next step (based on the lower entries of the $(5, 4)$ cell in Table 6.3).

Example 20. *Consider the case of Example 18. The LETS structures of irregular graphs in the class $(5, 4)$ are projected to normal graphs in classes $(5, 4)$, $(5, 6)$ and $(5, 8)$ of Table 6.3. One should thus apply $\{dot\}$, $\{dot\}$ and $\{dot, pa_2\}$ expansions to the instances of corresponding LETS structures, respectively.*

6.3 Efficient exhaustive Search Algorithm for LETSs of Irregular LDPC Codes

The search algorithm for LETSs of irregular graphs described in Section 6.2 can have two major problems, both occurring when the size of the characterization table for the variable-regular graphs with $d_v = t$, where t is the largest variable degree strictly less than a_{max}, happens to become too large. This can happen, in the cases that the value of t is relatively large (for example $t \geq 8$), or in the cases where the interest range of a is relatively large (for example $a_{max} \geq 10$), or in the cases where variable nodes with small degrees (for example, degree-2 variable nodes) are present. Under such circumstances, the range of a and b values covered in the characterization table of variable-regular graphs with $d_v = t$ plus the number of non-isomorphic LETS structures in the table will quickly increase. For example, there are more than nine million non-isomorphic LETS structures in the classes with $a = 10$ for variable-regular graphs with $d_v = 8$ ($|\mathcal{L}_8^{10}| = 9,545,887$). The first problem resulting from the large size of the characterization table is the excessive time that it takes the characterization algorithm of Chapter 5 to generate the table plus the large memory that is required to store the information of the characterization table. The second problem is the inefficiency of the search algorithm corresponding to a large characterization table, in the sense that, one may have to enumerate LETS structures with large values of b (and large multiplicities) that are not of direct interest but are parents of LETS structures of interest.

To overcome the above problems in circumstances just explained, we use a different approach to characterize/search LETSs of irregular codes. Rather than relying on the normal graph representation through the characterization table of a variable-regular graph, in the new approach, we focus on the class of LETS structures, i.e., rather than tracking the class of the normal graph within the characterization table of the corresponding variable-regular graph, we concern ourselves with the class of the LETS itself. For each class of LETS structures, we then identify *all* the possible expansions from the set of *dot*, *path* and *lollipop* expansions that can be applied to LETS structures in that class and eventually (after successive applications of one or more expansions) result in an (a, b) LETS structure with $a \leq a_{max}$ and $b \leq b_{max}$. As an example, consider the scenario of Example 18. Based on the previous approach, for any LETS structure of the irregular graphs which is projected to one of the two normal graphs in the $(5, 4)$ class of Table 6.3, regardless of the actual class of the LETS structure, the *dot* expansion is applied. In the new approach, however, the expansions are decided by the class of the LETS structure itself. For example, for a LETS structure in the $(5, 4)$ class, all the possible expansions that can eventually result in a LETS in the range of $a \leq 7$ and $b \leq 3$ are applied. These are *dot* and pa_2 expansions. (Note that based on Example 20, the LETS structures in the class of $(5, 4)$ can be projected into normal graphs in three different classes of Table 6.3, including the $(5, 4)$ class.) The application of this approach recursively and starting from all the simple cycles with size from $g/2$ up to a_{max} can provide us with an exhaustive list of all LETS structures of an irregular code within any desired range of $a \leq a_{max}$ and $b \leq b_{max}$. The indiscriminate application of this approach, however, is inefficient in both characterization and search of LETS structures. The reason is that many of the structures generated initially or in the intermediate stages of this process may not eventually reach any LETS structure within the range of interest. To overcome this problem while maintaining the exhaustiveness of the characterization/search, we identify the expansions for each class through a backward recursion. We start from LETS structures of size a_{max} with $b \leq b_{max}$, and then find out how these structures can be possibly constructed in a recursive fashion starting from simple cycles using the three expansions. The first step is to find out all the possible candidates that can lead to the target LETS structures through a single expansion step and then backtrack this recursively until one reaches simple cycles.

In the characterization/search of LETSs in variable-regular graphs, to find (cover)

all the LETSs in the range of $a \leq a_{max}$ and $b \leq b_{max}$, one should sometimes include (a, b) LETS structures with their b values outside the range of interest, i.e., $b_{max} < b \leq b'_{max}$. In Chapter 5, it was proved that the value of b'_{max} selected by the dpl characterization/search is minimal. In this work, for irregular graphs, for any given value of a in the range $g/2 \leq a \leq a_{max}$, we derive an upper bound b^a_{max}, on the b values for the classes of (a, b) LETS structures. This upper bound determines all the (a, b) classes of the same size a, i.e., (a, b) classes with $b \leq b^a_{max}$, that need to be covered in the characterization/search such that by the application of all the possible expansions to the included classes, starting from simple cycles, one can obtain all the LETS structures of the irregular graph within the range of interest $a \leq a_{max}$ and $b \leq b_{max}$, exhaustively. The upper bound is derived recursively by finding b^a_{max} as a function of b^{a+1}_{max} with the initial condition that $b^{a_{max}}_{max}$ is equal to b_{max}.

The following lemma imposes a limit on the maximum degree of a variable node that can be involved in a LETS structure given the interest range of $a \leq a_{max}$ and $b \leq b_{max}$. This will help to further constrain the space of search, and reduce its complexity.

Lemma 7. *Any variable node v involved in an (a, b) LETS structure of an irregular graph in the range of $a \leq a_{max}$ and $b \leq b_{max}$, has a degree $deg(v) < a_{max} + b_{max}$.*

Proof. Consider a variable node v with degree $deg(v) \geq a_{max} + b_{max}$ in an (a, b) LETS structure with $a \leq a_{max}$. This means that v is connected to at most $a_{max} - 1$ variable nodes within the LETS structure. Since $deg(v) \geq a_{max} + b_{max}$, this implies that v has at least $b_{max} + 1$ unsatisfied check nodes in its neighbourhood, which is in contradiction with the LETS structure having a b value of at most b_{max}.

■

As an intermediate step to deriving the upper bounds, in the following proposition, we first determine how the class of a LETS structure of an irregular graph changes as it is expanded by one of the three expansion techniques. In this proposition, we use the notation dot^k_m to denote a dot expansion with m edges and a root node with degree k.

Proposition 8. *Consider an (a, b) LETS structure $\mathcal{S}$ with variable node degrees $d_{v_i}, i = 1, \ldots, a$, in an irregular Tanner graph. The application of dot^k_m $(2 \leq m \leq k)$, pa_m $(m \geq 2)$, and lo^c_m $(m \geq g/2, g/2 \leq c \leq m)$ to $\mathcal{S}$ will result in LETS structure(s)*

in the $(a+1, b+k-2m)$, $(a+m, b-2+\sum_{i=1}^{m}(d_{v_i}-2))$, *and* $(a+m, b-2+\sum_{i=1}^{m}(d_{v_i}-2))$ *classes, respectively.*

Proof. Similar to the proof of Propositions 1, 3 and 4. ■

Corollary 1. *In an irregular Tanner graph with maximum variable degree* $d_{v_{max}}$ *and minimum variable degree* $d_{v_{min}} \geq 2$*, the application of* dot_m^k $(2 \leq m \leq k)$*,* pa_m $(m \geq 2)$*, and* lo_m^c $(m \geq g/2, g/2 \leq c \leq m)$ *to an* (a, b) *LETS structure results in an* (a', b') *LETS structure with minimum possible value of* b' *equal to* $(b - d_{v_{max}})^+$*,* $(b-2+m(d_{v_{min}}-2))^+$*, and* $(b-2+(m-1)(d_{v_{min}}-2)+d'-2)^+$*, respectively, where* $(f)^+ = max\{f, 0\}$*, and* d' *is equal to* $d_{v_{min}}$*, if* $d_{v_{min}} > 2$*, and is equal to the smallest variable degree strictly larger than* 2*, if* $d_{v_{min}} = 2$*. In particular, if* $d_{v_{min}} = 2$*, the minimum values of* b' *for* pa_m *and* lo_m^c *are* $(b-2)^+$ *and* $(b+d'-4)^+$*, respectively, where* d' *is equal to the smallest variable degree strictly larger than* 2*.*

Proof. For the dot_m^k expansion, the minimum value of b' is obtained when $m = k$ (note, $m \leq k$), and $m = d_{v_{max}}$. For the pa_m expansion, the minimum is attained when all the m nodes in the expansion have degree $d_{v_{min}}$. For the lo_m^c, the minimum is resulted when all the nodes in the expansion have degree $d_{v_{min}}$, with the exception being when $d_{v_{min}} = 2$, in which case, one node needs to have a degree equal to the smallest variable degree strictly larger than 2. ■

Based on Corollary 1, it can be seen that the *pa* and lo_m^c expansions can cause the decrease of at most 2 and 1 in the b value of a LETS structure, respectively. Since the minimum increase of a in *pa* and lo_m^c expansions is 2, successive increase of b_{max}^a by one will cover the range for *pa* and lo_m^c expansions. However, it can be seen in Corollary 1 that the *dot* expansion can cause the largest decrease in the b value of a LETS structure.

The following theorem establishes the upper bounds $b_{max}^a, g/2 \leq a \leq a_{max}$, through a recursive relationship.

Theorem 4. *Suppose that we are interested in generating all the* (a, b) *LETS structures of an irregular Tanner graph with variable node degree distribution* $\lambda(x)$ *and girth* g *within the range* $a \leq a_{max}$ *and* $b \leq b_{max}$*. For this, consider an approach that starts from simple cycles* $s_k, k = g/2, \ldots, a_{max}$*, and recursively applies all the possible dot, path and lollipop expansions to any generated LETS structure. Such an approach will exhaustively generate all the LETS structures in the range of interest,*

if for any size a in the range $g/2 \leq a \leq a_{max}$, the approach is constrained to only find (a, b) LETS structures with b values satisfying $b \leq b^a_{max}$, where b^a_{max} values are obtained through the recursion

$$b^a_{max} = b^{a+1}_{max} + max\{y, 2a - z\} \,. \tag{6.1}$$

In (6.1), y is the largest variable degree in $\lambda(x)$ less than or equal to a and z is the smallest variable degree in $\lambda(x)$ strictly larger than a and smaller than $a_{max} + b_{max}$. The initial condition for recursion is $b^{a_{max}}_{max} = b_{max}$.

Proof. Assume that b^{a+1}_{max} is obtained and we are looking to find b^a_{max}. The basis of the proof is to show that applying any dot^k_m to any LETS structures in an (a, b') class, where $b' > b^{a+1}_{max} + max\{y, 2a - z\}$, will result in LETSs in $(a + 1, b'')$ class, where $b'' > b^{a+1}_{max}$ and therefore we are not interested in. Based on Proposition 8, for dot^k_m expansion the most decrease of b value is $b' - b'' = 2m - k$. Suppose that there is a LETS structure, $\mathcal{S}$ in the (a, b') class, where $b' > b^{a+1}_{max} + max\{y, 2a - z\}$ and applying dot^k_m expansion to $\mathcal{S}$ results in a LETS in the $(a + 1, b'')$ class, where $b'' \leq b^{a+1}_{max}$. In the worst case, $b' = b^{a+1}_{max} + max\{y, 2a - z\} + 1$ and $b'' = b^{a+1}_{max}$. Therefore, $b^{a+1}_{max} + max\{y, 2a - z\} + 1 - b^{a+1}_{max} = 2m - k$ or $max\{y, 2a - z\} + 1 = 2m - k$. We want to show that there is no dot^k_m with any k and m values to satisfy this equality.

Value of k in dot^k_m is among the variable degrees in $\lambda(x)$ and $m \leq min\{a, k\}$. We consider two cases of $k \leq a$ and $k \geq a$. For the case of $k < a$: Based on Proposition 8, the most decrease for the b value is when k is maximized and $m = k$. This happens when $k = y$, the largest variable degree in $\lambda(x)$ less than or equal to a. The decrease of b is y. For the case of $k > a$: Since the Tanner graph free of 4-cycle are studied, $m \leq a$. Based on Proposition 8, the most decrease for the b value is when k is minimized and $m = a$. The minimum value of k is z, the smallest variable degree in $\lambda(x)$ strictly larger than a and smaller than $a_{max} + b_{max}$ (Lemma 14). The decrease of b is $2m - z = 2a - z$. Therefore the most decrease of b value for dot^k_m expansion $2m - k$ is at most $max\{y, 2a - z\}$ which is in contradiction.

The same happens for the cases of $b'' < b^{a+1}_{max}$ and $b' > b^{a+1}_{max} + max\{y, 2a - z\} + 1$. Since the smallest value of $max\{y, 2a - z\}$ is 2 (greater than 1 for pa_2), the bound caused by *dot* expansion is sufficient to find all the LETSs in the interest range. ■

We note that Theorem 4 provides a new dpl-based exhaustive characterization/search for LETS structures of irregular codes (compared to what was presented

in Section 6.2). We also note that the upper bounds derived based on Theorem 4 may not be necessarily tight, i.e., one may be able to find smaller bounds that still result in an exhaustive coverage of the LETS structures of interest, thus, further reducing the complexity of the search. One can also perform the search in a smaller space, by reducing the upper bounds, but possibly at the expense of sacrificing the exhaustiveness of the search. In fact, based on our experimental results, we propose the following upper bounds as a lower complexity alternative to (6.1):

$$b^a_{max} = b^{a+1}_{max} + max\{y, 2a - z\} - 2\,. \tag{6.2}$$

It is easy to see that (6.2) is smaller than (6.1). Although in our extensive simulations, presented in Section 6.5, the upper bounds (6.2) have always resulted in an exhaustive search, we have not been able to prove this, in general.

Given the upper bounds $b^{g/2}_{max}, \ldots, b^{a_{max}}_{max}$, Algorithm 13 provides a pseudo code for finding the list of expansion techniques that are required to be applied to all the non-isomorphic structures in each (a, b) LETS class. These expansions are stored in the (a, b) entry of table $\mathcal{EX}$, $\mathcal{EX}_{(a,b)}$. Based on Algorithm 13, the expansion *dot* can be applied to all the (a, b) classes with $a \leq a_{max} - 1$. Also, pa_m and lo^c_m can be applied to all the (a, b) classes with $a \leq a_{max} - m$. The only constraint for using an expansion technique is that the b value(s) of the new LETS structure(s) need to remain in the range identified by the upper bounds $b^{g/2}_{max}, \ldots, b^{a_{max}}_{max}$. The results of Corollary 1 are used to impose this constraint.

Example 21. *The outcome of Algorithm 13 is presented in Table 6.4 for the case of* $a_{max} \leq 7$, $b_{max} \leq 2$, *and an irregular graph with variable node degrees* $\{2, 3, 5, 10\}$ *and* $g = 6$. *These variable node degrees are used in [73] to design near-optimal irregular LDPC codes over binary-input additive white Gaussian noise (BIAWGN) channels. In Table 6.4, the notation dot is used to represent any* dot^k_m *expansion that results in* (a, b) *LETS structures with* $b \leq b^a_{max}$. *Also, the notation* lo_m *is used to represent all the* lo^c_m *expansion techniques with different values of* c. *Based on Theorem 4, we have* $b^7_{max} = 2$, $b^6_{max} = 7$, $b^5_{max} = 12$, $b^4_{max} = 15$, $b^3_{max} = 18$. *However, we note that based on Lemma 14, there is no LETS structure within the interest range that has a variable node with degree 10 or larger. Therefore in Proposition 2,* $d_{v_{max}} = 5$, *and* b^4_{max} *and* b^3_{max}, *as inputs of Algorithm 13 are modified to* $4(5-2) = 12$ *and* $3(5-2) = 9$,

Algorithm 3 Finding the expansion techniques $\mathcal{EX}_{(a,b)}$ for the (a, b) classes of LETS structures, $g/2 \leq a \leq a_{max}$, $1 \leq b \leq b_{max}$, for an irregular Tanner graph with girth g.

Inputs: $a_{max}, b_{max}^{g/2}, \ldots, b_{max}^{a_{max}} = b_{max}, g$.
Initializations: $a = a_{max} - 1$.
while $a \geq g/2$ **do**
    $\mathcal{EX}_{(a,b)} \leftarrow dot, \ \forall \ b \in \{2, \ldots, b_{max}^{a}\}$.
    for $b = 1, \ldots, b_{max}^{a}$ **do**
        $m = 2$.
        while $a + m \leq a_{max}$ **do**
            if $0 \leq b - 2 \leq b_{max}^{a+m}$ **then**
                $\mathcal{EX}_{(a,b)} \leftarrow pa_m$.
            end if
            if $m \geq g/2$ and $b - 1 \leq b_{max}^{a+m}$ **then**
                for $c = g/2, \ldots, m$ **do**
                    $\mathcal{EX}_{(a,b)} \leftarrow lo_m^c$.
                end for
            end if
            $m = m + 1$.
        end while
    end for
    $a = a - 1$.
end while
Output: $\mathcal{EX}$.

Table 6.4: Expansions Required for (a, b) Classes of Irregular Graphs with Variable Degrees $2, 3, 5, 10$, and $g = 6$ for $a \leq a_{max} = 7$ and $b \leq b_{max} = 2$

	$a = 3$	$a = 4$	$a = 5$	$a = 6$	$a = 7$
b = 0	-	-	-	-	-
b = 1	lo_3, lo_4	lo_3	-	-	-
b = 2	$dot, pa_2, pa_3, pa_4, lo_3, lo_4$	dot, pa_2, pa_3, lo_3	dot, pa_2	dot	-
b = 3	$dot, pa_2, pa_3, pa_4, lo_3, lo_4$	dot, pa_2, pa_3, lo_3	dot, pa_2	dot	
b = 4	$dot, pa_2, pa_3, pa_4, lo_3$	dot, pa_2, pa_3	dot, pa_2	dot	
b = 5	dot, pa_2, pa_3, lo_3	dot, pa_2	dot	dot	
b = 6	dot, pa_2, pa_3, lo_3	dot, pa_2	dot	dot	
b = 7	dot, pa_2, pa_3, lo_3	dot, pa_2	dot	dot	
b = 8	dot, pa_2, pa_3, lo_3	dot, pa_2	dot		
b = 9	dot, pa_2, pa_3	dot, pa_2	dot		
b = 10		dot	dot		
b = 11		dot	dot		
b = 12		dot	dot		

respectively.

The pseudo code of the proposed search algorithm is given in Algorithm 4. Having the upper bounds $b_{max}^{g/2}, \ldots, b_{max}^{a_{max}}$, and the expansion table $\mathcal{EX}$, as the input, the search algorithm starts by the enumeration of simple cycles of length up to a_{max} through Routine 9. Note that for any s_k, where $g/2 \leq k \leq a_{max}$, the number of unsatisfied check nodes of the instances should be less than or equal b_{max}^{k}, i.e., only simple cycles that satisfy this condition are stored for further processing. This is performed in Lines 12-14 of Routine 9. After the enumeration of all instances of a simple cycle, these instances are expanded in the while loop (Lines 9-31) to find instances of other LETS structures in the interest range. The selected expansions for each class are those stored in $\mathcal{EX}$. In Algorithm 4, the notation $\mathcal{I}_k^{a,b}$ is used for the set of LETS instances in the (a, b) class found by starting from the instances of the simple cycle of length k, and $\mathcal{I}$ is the set of all instances which are found so far in the algorithm. Finding the instances of LETS structures using *dot*, *path* and *lollipop* expansion techniques are explained in Routines 10, 11 and 12, respectively.

The complexity of the search algorithm depends, in general, on the multiplicity of different instances of LETS structures and the expansion techniques used in different classes. A detailed discussion on the complexity of the dpl-based search algorithm for variable-regular graphs can be found in Chapter 5. The generalization of those discussions to irregular graphs is rather simple and thus not presented here.

6.4 Efficient Exhaustive Search of Elementary Trapping sets for Irregular LDPC Codes

Leafless ETSs (LETSs) are known to be the main problematic structures in the error floor region of variable-regular LDPC codes. For irregular LDPC codes, however, in addition to LETSs, there are other ETSs that are problematic but are not leafless, i.e., they have variable nodes that are connected to only one satisfied check node [62], [1], [47], [53], [76], [23]. This is particularly the case for irregular codes with degree-2 variable nodes [47], [53]. In Sections 6.2 and 6.3, we studied the LETS structures of

Algorithm 4 (LETS Exhaustive Search) Finds all the instances of (a, b) LETS structures of an irregular Tanner graph G with girth g, for $a \leq a_{max}$ and $b \leq b_{max}$. The inputs are the upper bounds $b_{max}^{g/2}, \ldots, b_{max}^{a_{max}}$, and expansion techniques $\mathcal{EX}$ provided by Algorithm 13. The output is the set $\mathcal{I}$, which contains all the instances of LETS structures in the interest range.

Inputs: $G, a_{max}, \{b_{max}^{g/2}, \ldots, b_{max}^{a_{max}}\}, \mathcal{EX}, g$.
Initializations: $\mathcal{I} \leftarrow \emptyset$.
for $k = g/2, \ldots, a_{max}$ **do**
    $\mathcal{I}_k = \{\mathcal{I}_k^{k,0}, \ldots, \mathcal{I}_k^{k,b_{max}^k}\}$= **CycSrch**$(G, k)$.
    $\mathcal{I} = \mathcal{I} \cup \mathcal{I}_k$.
end for
for $k = g/2, \ldots, a_{max}$ **do**
    $a = k$.
    while $a < a_{max}$ **do**
        for $b = 1, \ldots, b_{max}^a$ **do**
            if $dot \in \mathcal{EX}_{(a,b)}$ **then**
                $\{\mathcal{I}_{tem}^{a+1,0}, \ldots, \mathcal{I}_{tem}^{a+1,b_{max}^{a+1}}\}$= **DotSrch**$(G, \mathcal{I}_k^{a,b}, \mathcal{I})$.
            end if
            for any $pa_m \in \mathcal{EX}_{(a,b)}$ **do**
                $\{\mathcal{I}_{tem}^{a+m,0}, \ldots, \mathcal{I}_{tem}^{a+m,b_{max}^{a+m}}\}$= **PathSrch**$(G, \mathcal{I}_k^{a,b}, \mathcal{I}, m)$.
            end for
            for any $lo_m^c \in \mathcal{EX}_{(a,b)}$ **do**
                $\{\mathcal{II}_{tem}^{a+m,0}, \ldots, \mathcal{II}_{tem}^{a+m,b_{max}^{a+m}}\}$= **LolliSrch**$(G, \mathcal{I}_k^{a,b}, \mathcal{I}, \mathcal{I}_c, m)$.
                for $s = 1, \ldots, b_{max}^a$ **do**
                    $\mathcal{I}_{tem}^{a+m,s} = \mathcal{I}_{tem}^{a+m,s} \cup \mathcal{II}_{tem}^{a+m,s}$.
                end for
            end for
            for $t = a + 1, \ldots, a_{max}$ **do**
                for $s = 1, \ldots, b_{max}^t$ **do**
                    $\mathcal{I}_k^{t,s} = \mathcal{I}_k^{t,s} \cup \mathcal{I}_{tem}^{t,s}$.
                    $\mathcal{I} = \mathcal{I} \cup \mathcal{I}_{tem}^{t,s}$.
                end for
            end for
        end for
        $a = a + 1$.
    end while
end for
Output: $\mathcal{I}$.

Routine 9 (CycSrch) Finds all the instances of simple cycles of length k with $b \leq b_{max}^k$, in a given Tanner graph G. $\{\mathcal{I}_k^{k,0}, \ldots, \mathcal{I}_k^{k,b_{max}^k}\}$= CycSrch$(G, k)$

Initializations: $\mathcal{I}_k^{k,b} \leftarrow \emptyset,\ \forall\ b \leq b_{max}^k$.
for each variable node v_l in G **do**
  for each check node c_i in the neighbourhood of v_l **do**
    Find all the paths $\mathcal{PA}_{i,l}$ of length $k-1$ in G, starting from c_i that do not contain v_l.
  end for
  for any pair of check nodes c_i and c_j in the neighbourhood of v_l, where $i \neq j$ **do**
    for any path $pa \in \mathcal{PA}_{i,l}$, and any path $pa' \in \mathcal{PA}_{j,l}$ **do**
      if the two paths end with the same node and that node is their only common node **do**
        Let v and v' denote the last variable nodes of pa and pa', respectively.
        if variable nodes in $pa \setminus v$ and $pa' \setminus v'$ do not have any common check node **do**
          $\mathcal{S} = v_l \cup \{$set of variable nodes in $pa \cup pa'\}$.
          if $|\Gamma_o(\mathcal{S})| \leq b_{max}^k$ **then**
            $\mathcal{I}_k^{k,|\Gamma_o(\mathcal{S})|} = \mathcal{I}_k^{k,|\Gamma_o(\mathcal{S})|} \cup \{\mathcal{S}\}$.
          end if
        end if
      end if
    end for
  end for
end for
Outputs: $\{\mathcal{I}_k^{k,0}, \ldots, \mathcal{I}_k^{k,b_{max}^k}\}$.

Routine 10 (DotSrch) Expanding all the instances of LETS structures in the (a, b) class $\mathcal{I}_k^{a,b}$ of Tanner graph G using *dot* expansions to find a set of instances of LETS structures of size $a+1$, excluding the already found structures $\mathcal{I}$, and storing the rest in $\{\mathcal{I}_{tem}^{a+1,0}, \ldots, \mathcal{I}_{tem}^{a+1,b_{max}^{a+1}}\}$. $\{\mathcal{I}_{tem}^{a+1,0}, \ldots, \mathcal{I}_{tem}^{a+1,b_{max}^{a+1}}\}$= DotSrch $(G, \mathcal{I}_k^{a,b}, \mathcal{I})$

Initializations: $\mathcal{I}_{tem}^{a+1,b} \leftarrow \emptyset,\ \forall\ b \leq b_{max}^{a+1}$.
for each instance of LETS structure $\mathcal{S}$ in $\mathcal{I}_k^{a,b}$ **do**
    Consider $\mathcal{V}$ to be the set of variable nodes in $V \setminus \mathcal{S}$, which have at least two connections with the check nodes in $\Gamma_o(\mathcal{S})$ and have no connection with the check nodes in $\Gamma_e(\mathcal{S})$.
    for each variable node $v \in \mathcal{V}$ **do**
        $\mathcal{S}' = \mathcal{S} \cup v$.
        if $|\Gamma_o(\mathcal{S}')| \leq b_{max}^{a+1}$ **then**
            $\mathcal{I}_{tem}^{a+1,|\Gamma_o(\mathcal{S}')|} = \mathcal{I}_{tem}^{a+1,|\Gamma_o(\mathcal{S}')|} \cup \{\mathcal{S}' \setminus \mathcal{I}\}$.
        end if
    end for
end for
Output: $\{\mathcal{I}_{tem}^{a+1,0}, \ldots, \mathcal{I}_{tem}^{a+1,b_{max}^{a+1}}\}$.

irregular LDPC codes. This section is dedicated to ETSs of irregular codes that have leaves. We use the notation "ETSL" for such trapping sets, and remind the reader that it is the the normal graph representation of ETSL structures that has at least one leaf. Two examples of ETSL structures in the $(3, 3)$ and $(5, 4)$ classes, along with their normal graphs, are shown in Fig. 6.5.

The *depth-one tree (dot)* expansion technique plays an important role in the characterization and search of ETSLs. However, unlike the LETS case, where dot_m^k expansion with $m \geq 2$ was used, in the ETSL case, we are interested in the dot_m^k expansion

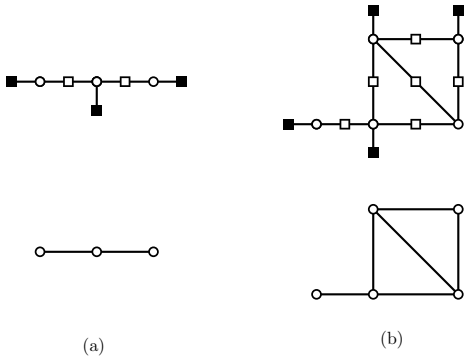


Figure 6.5: Two ETSLs in the (a) $(3, 3)$ and (b) $(5, 4)$ classes, respectively.

Routine 11 (PathSrch) Expanding all the instances $\mathcal{I}_k^{a,b}$ of LETS structures in the (a, b) class of Tanner graph G using pa_m to find instances of LETS structures of size $a + m$, excluding the already found instances $\mathcal{I}$, and storing the rest in $\{\mathcal{I}_{tem}^{a+m,0}, \ldots, \mathcal{I}_{tem}^{a+m,b_{max}^{a+m}}\}$. $\{\mathcal{I}_{tem}^{a+m,0}, \ldots, \mathcal{I}_{tem}^{a+m,b_{max}^{a+m}}\}$= PathSrch$(G, \mathcal{I}_k^{a,b}, \mathcal{I}, m)$

Initializations: $\mathcal{I}_{tem}^{a+m,b} \leftarrow \emptyset,\ \forall\ b \leq b_{max}^{a+m}$.
for each LETS instance $\mathcal{S}$ in $\mathcal{I}_k^{a,b}$ **do**
    for each unsatisfied check node $c_k \in \Gamma_o(\mathcal{S})$ **do**
        Find all the paths $\mathcal{PA}_k$ of length m in the Tanner graph, starting from c_k.
    end for
    for any pair of unsatisfied check nodes c_k and c_j in $\Gamma_o(\mathcal{S})$, where $k \neq j$ **do**
        for any path $pa \in \mathcal{PA}_k$ and any path $pa' \in \mathcal{PA}_j$ **do**
            if pa and pa' end with the same node and this node is their only common node, and if variable and check nodes of pa and pa' are not in $G(\mathcal{S})$ **do**
                $\mathcal{S}' = \mathcal{S} \cup \{$set of variable nodes in $pa \cup pa'\}$.
                if $|\Gamma_o(\mathcal{S}')| \leq b_{max}^{a+m}$ **then**
                    $\mathcal{I}_{tem}^{a+m,|\Gamma_o(\mathcal{S}')|} = \mathcal{I}_{tem}^{a+m,|\Gamma_o(\mathcal{S}')|} \cup \{\mathcal{S}' \setminus \mathcal{I}\}$.
                end if
            end if
        end for
    end for
end for
Outputs: $\{\mathcal{I}_{tem}^{a+m,0}, \ldots, \mathcal{I}_{tem}^{a+m,b_{max}^{a+m}}\}$.

Routine 12 (LolliSrch) Expanding all the instances $\mathcal{I}_k^{a,b}$ of LETS structures in the (a, b) class of Tanner graph G using lo_m^c to find instances of LETS structures of size $a + m$, excluding the already found instances $\mathcal{I}$, and storing the rest in $\{\mathcal{I}_{tem}^{a+m,0}, \ldots, \mathcal{I}_{tem}^{a+m,b_{max}^{a+m}}\}$. The set of instances of simple cycles of length c, $\mathcal{I}_c$, is also an input. $\{\mathcal{I}_{tem}^{a+m,0}, \ldots, \mathcal{I}_{tem}^{a+m,b_{max}^{a+m}}\}$= LolliSrch $(G, \mathcal{I}_k^{a,b}, \mathcal{I}, \mathcal{I}_c, m)$

Initializations: $\mathcal{I}_{tem}^{a+m,b} \leftarrow \emptyset,\ \forall\ b \leq b_{max}^{a+m}$, $d = m + 1 - c$.
for each LETS instance $\mathcal{S}$ in $\mathcal{I}_k^{a,b}$ **do**
    Find all the paths $\mathcal{PA}$ of length $2(d-1)$ in the Tanner graph, starting from check nodes c' in $\Gamma_o(\mathcal{S})$ that have no common nodes with $G(\mathcal{S})$ other than c'.
    for each structure $\mathcal{C} \in \mathcal{I}_c$, for which $G(\mathcal{C})$ has no common node with $G(\mathcal{S})$, let $\Gamma_o(\mathcal{C})$ denote the set of unsatisfied check nodes of $G(\mathcal{C})$ **do**
        for each path, $pa \in \mathcal{PA}$ **do**
            if pa ends with a check node c'' in $\Gamma_o(\mathcal{C})$ and if c'' is the only common node between pa and $\Gamma_o(\mathcal{C})$ **do**
                $\mathcal{S}' = \{\mathcal{S} \cup \{\text{set of variable nodes in } pa \cup G(\mathcal{C})\}$.
                if $|\Gamma_o(\mathcal{S}')| \leq b_{max}^{a+m}$ **then**
                    $\mathcal{I}_{tem}^{a+m,|\Gamma_o(\mathcal{S}')|} = \mathcal{I}_{tem}^{a+m,|\Gamma_o(\mathcal{S}')|} \cup \{\mathcal{S}' \setminus \mathcal{I}\}$.
                end if
            end if
        end for
    end for
end for
Output: $\{\mathcal{I}_{tem}^{a+m,0}, \ldots, \mathcal{I}_{tem}^{a+m,b_{max}^{a+m}}\}$.

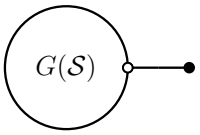


Figure 6.6: Expansion of an ETS structure $\mathcal{S}$ with dot_1^k.

with $m = 1$. Fig. 6.6 shows a structure expanded by dot_1^k.

Lemma 8. *Suppose that $\mathcal{S}$ is an (a, b) ETS structure of irregular Tanner graphs. Expansion of $\mathcal{S}$ using dot_1^k, results in ETS structure(s) in the $(a+1, b+k-2)$ class, where k is the degree of the new variable node added to $\mathcal{S}$.*

In general, the ETSL structures can be partitioned into two categories. The ETSLs that contain at least one LETS sub-structure, and those that do not contain any LETS sub-structure. We use notations ETSL_1 and ETSL_2, to represent these two categories, respectively. The structures in Fig. 6.5 (a) and (b) are examples of ETSL_2 and ETSL_1, respectively. In the rest of this section, ETSL_1 and ETSL_2 structures are characterized and an efficient search algorithm is presented to find them in an exhaustive fashion.

6.4.1 Characterization of ETSL_1 and ETSL_2

We characterize ETSL_1 structures as the expansions of their LETS sub-structures through the following proposition.

Proposition 9. *Any (a, b) $ETSL_1$ structure $\mathcal{S}$ of irregular graphs, with minimum variable node degree $d_{v_{min}} \geq 2$, in the range of $a \leq a_{max}$ and $b \leq b_{max}$, can be obtained by successive applications of dot_1^k to the largest (a', b') LETS substructure of $\mathcal{S}$, where $a' \leq a_{max}$ and $b' \leq b_{max}$.*

Proof. Suppose that $\mathcal{S} = (F, E)$, and $\mathcal{S}' = (F', E')$ is the largest LETS sub-structure of $\mathcal{S}$. We assume that the expansion $F \backslash F'$ is connected. Disconnected expansions can be treated as a collection of connected expansions. We first prove that the number of edges connecting $F \backslash F'$ to the nodes in $\mathcal{S}'$ is one. Suppose that the number of such edges is at least 2. If at least two of these edges are connected to one node, v, in $F \backslash F'$, then adding v to $\mathcal{S}'$ results in a LETS sub-structure of $\mathcal{S}$ that is larger than $\mathcal{S}'$. This is a contradiction. So the connecting edges must be each connected to a

distinct node in $F \backslash F'$. Consider two such nodes and call them v_1 and v_2. Since the expansion is connected, there is at least one path between v_1 and v_2 in the expansion. One can thus add all the nodes on that path plus v_1 and v_2 to $\mathcal{S}'$, and obtain a LETS substructure of $\mathcal{S}$ larger than $\mathcal{S}'$. Again, a contradiction. We thus conclude that the expansion is connected by only one edge to $\mathcal{S}'$. (Suppose that the node in $F \backslash F'$ which is connected to $\mathcal{S}'$ is v.) With a similar approach, it can be shown that the expansion can not contain a cycle. Since the expansion has only one connection to $\mathcal{S}'$ and does not contain a cycle, the only possibility is a rooted tree with the root at node v. It is now easy to see that a rooted tree, as an expansion, can be implemented by successive applications of dot_1^k expansions. Using Lemma 8, one can see that as dot_1^k expansions with $k \geq 2$ are applied to the LETS structure $\mathcal{S}'$, by each application, the a value is increased by one, and the b value is either increased or remains the same. This implies that if for the target (a, b) ETSL_1 structure $\mathcal{S}$, we have $a \leq a_{max}$ and $b \leq b_{max}$, then for the largest (a', b') LETS substructure of $\mathcal{S}$, we must have $a' \leq a_{max}$ and $b' \leq b_{max}$. ■

Based on Proposition 12, to find all the ETSL_1s of an irregular LDPC code in the interest range of $a \leq a_{max}$ and $b \leq b_{max}$, one can first find all the instances of (a, b) LETS structures with $a \leq a_{max}$ and $b \leq b_{max}$, and then apply dot_1^k expansions successively to these instances, as appropriate. One should note that all the instances generated in the process of the application of dot_1^k expansions to a LETS are ETSL_1s. The following corollary determines which dot_1^k expansion should be applied to which LETS or ETSL_1 instance.

Corollary 2. *To generate all the (a, b) $ETSL_1$ instances of an irregular graph with $d_{v_{min}} \geq 2$, in the range of $a \leq a_{max}$ and $b \leq b_{max}$, for each value of i in the range $d_{v_{min}} - 2 \leq i \leq b_{max} - 1$, the expansion dot_1^k with $k \leq i + 2$ should be applied to any LETS and $ETSL_1$s with $a \leq a_{max} - 1$ and $b = b_{max} - i$.*

Example 22. *Based on Corollary 2, if one is interested in finding all $ETSL_1$s in the interest range of $a \leq a_{max}$ and $b \leq 2$, for an irregular graph with $d_{v_{min}} = 2$, expansion dot_1^2 should be applied to LETSs and $ETSL_1$s with $a \leq a_{max} - 1$ and $b = 2$, and dot_1^2 and dot_1^3 expansions should be applied to LETSs and $ETSL_1$s with $a \leq a_{max} - 1$ and $b = 1$.*

Remark 7. *Although having variable nodes of degree one is not common in most LDPC codes, the results presented here can easily be generalized to the case where the*

code has such variable nodes.

It is easy to see that ETSL_2 structures (in the space of normal graphs) contain no cycles, and are thus trees. The following proposition is then simple to prove.

Proposition 10. *Any (a, b) $ETSL_2$ structure of irregular graphs with $d_{v_{min}} \geq 2$, in the range $a \leq a_{max}$ and $b \leq b_{max}$, can be obtained by successive applications of dot_1^k expansions to single variable nodes with degree less than or equal to b_{max}.*

Based on Proposition 10, for the special case of $b_{max} = 2$, the only configuration of ETSL_2 is a chain consisting of only degree-2 variable nodes.

6.4.2 Exhaustive Search of ETSLs in Irregular Tanner Graphs

Based on the results of Subsection 6.4.1, Algorithm 5 provides a pseudo code for finding all the instances of all the (a, b) ETSL structures of an irregular Tanner graph in the interest range of $a \leq a_{max}$ and $b \leq b_{max}$, exhaustively.

We note that the complexity of finding ETSLs is often negligible (less than 1 %) in comparison with the complexity of search for LETSs.

6.5 Numerical Results

We have applied the proposed search algorithm of Section 6.3 (for LETSs) and that of Section 6.4 (for ETSL) to find the ETSs of a large number of irregular LDPC codes with a wide range of variable node degrees, rates and block lengths, exhaustively. These codes and their parameters including block length, rate, girth and variable and check node degree distributions are listed in Table 6.5. For each code, we have used both the upper bounds of (6.1) and (6.2), and observed that, although we have no proof that the latter will result in an exhaustive search in general, for all the codes tested in this work, both upper bounds provide an exhaustive coverage of LETSs.

Except codes $\mathcal{C}_1$-$\mathcal{C}_6$, $\mathcal{C}_{17}$, and $\mathcal{C}_{18}$, the other LDPC codes are all structured codes. Codes $\mathcal{C}_7$-$\mathcal{C}_{15}$ are the LDPC codes used in the IEEE 802.16e standard [101], and Code $\mathcal{C}_{16}$ is a code used in the IEEE 802.11n [100]. For structured codes, their structural properties are used to simplify the search.

Algorithm 5 (ETSL Exhaustive Search) Finds all the instances of (a, b) ETSL structures of an irregular Tanner graph $G = (V, E)$ with girth g and $d_{v_{min}} \geq 2$, for $a \leq a_{max}$ and $b \leq b_{max}$. The input is all the instances of (a, b) LETS structures, in the range $a \leq a_{max}$ and $b \leq b_{max}$, $\mathcal{I}$. ($\mathcal{I}^{a,b}$ is the set of all the instances in the (a, b) class.) The output is the set $\mathcal{I}_{ETSL}$, which contains all the instances of ETSL_1s and ETSL_2s in the interest range.

Inputs: $G, a_{max}, \mathcal{I}, g$.
Initializations: $\mathcal{I}_{tem}^{a,b} \leftarrow \emptyset, \mathcal{I}_{ETSL}^{a,b} \leftarrow \emptyset, \ \forall\ a \leq a_{max}, \ \forall\ b \leq b_{max}$.
for $b = 2, \ldots, b_{max}$ **do**
    $\mathcal{I}_{tem}^{1,b}$ = the set of variable nodes of degree b in G.
end for
for $a = 1, \ldots, a_{max} - 1$ **do**
    for $b = 1, \ldots, b_{max}$ **do**
        $\mathcal{I}_{tem}^{a,b} = \mathcal{I}_{tem}^{a,b} \cup \mathcal{I}^{a,b}$.
        for any structure $\mathcal{S} \in \mathcal{I}_{tem}^{a,b}$ **do**
            Consider $\mathcal{V}$ to be the set of variable nodes in $V \setminus \mathcal{S}$ with degrees less than or equal to $b_{max} + 2 - b$, which have only one connection to the check nodes in $\Gamma_o(\mathcal{S})$ and have no connection to the check nodes in $\Gamma_e(\mathcal{S})$,
            for each variable node $v \in \mathcal{V}$ **do**
                $\mathcal{I}_{ETSL}^{a+1,b'} = \mathcal{I}_{ETSL}^{a+1,b'} \cup \{\mathcal{S} \cup v\}$, $\mathcal{I}_{tem}^{a+1,b'} = \mathcal{I}_{tem}^{a+1,b'} \cup \{\mathcal{S} \cup v\}$, where $b' = b + deg(v) - 2$.
            end for
        end for
    end for
end for
Output: $\mathcal{I}_{ETSL} = \{\mathcal{I}_{ETSL}^{a,b}, \ \forall\ a \leq a_{max}, \ \forall\ b \leq b_{max}\}$.

Table 6.5: List of irregular LDPC Codes Used in This Chapter

Code	n	R	g	$\lambda(x)$	$\rho(x)$	Ref
$\mathcal{C}_1$	1000	0.7	6	$\lambda(x) = 0.243x^2 + 0.757x^3$	$\rho(x) = 0.002x^7 + 0.012x^8 + 0.043x^9 + 0.151x^{10} +$ $0.318x^{11} + 0.264x^{12} + 0.144x^{13} + 0.052x^{14} + 0.014x^{15}$	[65]
$\mathcal{C}_2$	8000	0.78	6	$\lambda(x) = 0.636x^2 + 0.364x^3$	$\rho(x) = 0.001x^{11} + 0.020x^{12} + 0.345x^{13}$ $+0.557x^{14} + 0.068x^{15} + 0.008x^{16} + 0.001x^{17}$	
$\mathcal{C}_3$	4000	0.5	6	$\lambda(x) = 0.871x^2 + 0.129x^3$	$\rho(x) = 0.008x^4 + 0.757x^5 + 0.231x^6 + 0.004x^7$	
$\mathcal{C}_4$	500	0.5	6	$\lambda(x) = 0.636x^2 + 0.364x^3$	$\rho(x) = 0.030x^4 + 0.360x^5 + 0.530x^6 + 0.063x^7 + 0.017x^8$	
$\mathcal{C}_5$	1000	0.5	6	$\lambda(x) = 0.762x^3 + 0.238x^4$	$\rho(x) = 0.008x^5 + 0.062x^6 + 0.480x^7$ $+0.358x^8 + 0.079x^9 + 0.01x^{10} + 0.003x^{11}$	
$\mathcal{C}_6$	300	0.5	6	$\lambda(x) = 0.878x^3 + 0.122x^4$	$\rho(x) = 0.019x^3 + 0.110x^4 + 0.522x^5$ $+0.302x^6 + 0.039x^7 + 0.008x^9$	
$\mathcal{C}_7$	576	0.75	6	$\lambda(x) = 0.114x + 0.409x^2 + 0.477x^5$	$\rho(x) = 0.318x^{13} + 0.682x^{14}$	[101]
$\mathcal{C}_8$	1056	0.75	6			
$\mathcal{C}_9$	2304	0.75	6			
$\mathcal{C}_{10}$	576	0.66	6	$\lambda(x) = 0.173x + 0.037x^2 + 0.790x^3$	$\rho(x) = 0.864x^9 + 0.136x^{10}$	
$\mathcal{C}_{11}$	1056	0.66	6			
$\mathcal{C}_{12}$	2304	0.66	6			
$\mathcal{C}_{13}$	576	0.5	6	$\lambda(x) = 0.289x + 0.316x^2 + 0.395x^5$	$\rho(x) = 0.632x^5 + 0.368x^6$	
$\mathcal{C}_{14}$	1056	0.5	6			
$\mathcal{C}_{15}$	2304	0.5	6			
$\mathcal{C}_{16}$	1944	0.5	6	$\lambda(x) = 0.256x + 0.314x^2$ $+0.046x^3 + 0.384x^{10}$	$\rho(x) = 0.814x^6 + 0.186x^7$	[100]
$\mathcal{C}_{17}$	504	0.5	8	$\lambda(x) = 0.239x + 0.210x^2 + 0.036x^3 +$ $0.122x^4 + 0.014x^6 + 0.007x^{13} + 0.372x^{14}$	$\rho(x) = 0.077x^6 + 0.834x^7 + 0.089x^8$	[57]
$\mathcal{C}_{18}$	1008	0.5	8	$\lambda(x) = 0.239x + 0.210x^2 + 0.035x^3 +$ $0.121x^4 + 0.016x^6 + 0.003x^{13} + 0.376x^{14}$	$\rho(x) = 0.009x^6 + 0.978x^7 + 0.013x^8$	

Table 6.6: Multiplicities of (a, b) ETSs, LETSs, EASs and FEASs of Codes $\mathcal{C}_1$ and $\mathcal{C}_2$ within the range of $a \leq 7$ and $b \leq 3$

	$\mathcal{C}_1$				$\mathcal{C}_2$			
(a, b) class	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS	Total LETS	Total EAS	Total FEAS
(3,3)	32	32	32	23	904	904	904	854
(4,2)	1	1	1	1	13	13	13	13
(4,3)	19	19	10	8	52	52	27	25
(5,2)	0	0	0	0	2	2	2	2
(5,3)	28	27	23	21	758	540	540	508
(6,2)	1	1	1	1	13	13	13	12
(6,3)	34	34	30	18	127	93	77	70
(7,1)	0	0	0	0	1	1	1	1
(7,2)	1	1	1	1	4	4	4	4
(7,3)	79	70	56	43	777	541	538	506
Search w. (6.1)	33 min.				36 min.			
Search w. (6.2)	13 min.				29 min.			

Tables 6.6-6.12 list the multiplicity of instances of ETSs, LETSs, EASs and FEASs in different (a, b) classes, $a \leq a_{max}, b \leq b_{max}$, for these codes. Each row of a table corresponds to a non-empty ETS class, and for each class, the total number of instances of ETSs, LETSs, EASs and FEASs are listed. The difference between the total number of ETSs and LETSs gives the total number of ETSLs. In the last two rows of each table, the run-times of the search algorithm based on the upper bounds of (6.1) and (6.2) are reported, respectively. Comparison of the run-times shows that large improvements in the search speed, in some cases more than an order of magnitude, can be obtained by using (6.2) instead of (6.1).

Remark 8. *Since the variable degrees of $\mathcal{C}_1$ are 3 and 4, and we are interest to find the ETSs in the small range of $a \leq 7$ and $b \leq 3$, it takes the search algorithm proposed in Section 6.2 21 minutes to find all the structures reported in Table 6.6 for $\mathcal{C}_1$. It is based on the characterization Table 6.3. However, as mentioned before, this approach is not efficient for the irregular codes with large variable degrees and large interest range of a and b. For example, for the code $\mathcal{C}_{16}$ with variable degrees $\{2, 3, 4, 11\}$, to find all the LETSs in the interest range of $a \leq 12$ and $b \leq 2$ (Table 6.11), based on Theorem 3, one should use the characterization of variable-regular graph with $d_v = 11$, in the range of $a \leq 11$ and $b \leq 110$!*

Table 6.7: Multiplicities of (a, b) ETSs, LETSs, ETSs and FEASs of Codes $\mathcal{C}_3$ and $\mathcal{C}_4$ within the range of $a \leq 9$ and $b \leq 4$

	$\mathcal{C}_3$				$\mathcal{C}_4$			
(a, b) class	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS	Total LETS	Total EAS	Total FEAS
(2,4)	24610	0	0	0	1938	0	0	0
(3,3)	110	110	110	110	62	62	62	58
(3,4)	96	96	0	0	169	169	0	0
(4,3)	1	1	1	1	13	13	9	9
(4,4)	2336	826	826	804	1077	397	386	303
(5,3)	13	13	13	11	42	42	42	37
(5,4)	38	26	16	15	375	243	161	126
(6,2)	1	1	1	1	1	1	1	1
(6,3)	1	1	1	1	18	18	15	14
(6,4)	409	231	231	221	1145	680	641	502
(7,2)	0	0	0	0	1	1	1	1
(7,3)	15	6	6	6	54	46	44	37
(7,4)	25	10	8	8	762	579	473	357
(8,2)	0	0	0	0	4	4	4	3
(8,3)	0	0	0	0	47	39	39	35
(8,4)	249	72	72	70	1831	1283	1203	918
(9,1)	0	0	0	0	1	1	1	1
(9,3)	3	3	3	2	110	81	78	62
(9,4)	7	7	5	5	2001	1521	1331	1015
Search w. (6.1)	16 min.				57 min.			
Search w. (6.2)	15 min.				55 min.			

Table 6.8: Multiplicities of (a, b) ETSs, LETSs, EASs and FEASs of Codes $\mathcal{C}_5$ and $\mathcal{C}_6$ within the range of $a \leq 8$ and $b \leq 7$

	$\mathcal{C}_5$				$\mathcal{C}_6$			
(a, b) class	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS	Total LETS	Total EAS	Total FEAS
(2,6)	9146	0	0	0	2516	0	0	0
(2,7)	5673	0	0	0	672	0	0	0
(3,6)	830	830	0	0	486	486	0	0
(3,7)	1034	1034	0	0	256	256	0	0
(4,4)	0	0	0	0	1	1	1	1
(4,6)	150	150	0	0	180	180	0	0
(4,7)	275	275	0	0	168	168	0	0
(5,6)	35	35	1	1	110	93	0	0
(5,7)	121	121	0	0	149	146	0	0
(6,6)	15	15	1	1	69	69	8	4
(6,7)	52	52	2	0	131	131	5	1
(7,6)	5	5	0	0	49	49	6	2
(7,7)	23	23	3	1	133	133	14	1
(8,5)	0	0	0	0	2	2	2	1
(8,6)	0	0	0	0	40	40	9	3
(8,7)	13	13	2	1	151	151	20	7
Search w. (6.1)	219 min.				42 min.			
Search w. (6.2)	15 min.				6 min.			

Table 6.9: Multiplicities of (a, b) ETSs, LETSs, EASs and FEASs of Codes $\mathcal{C}_7$-$\mathcal{C}_{12}$ within the range of $a \leq 8$ and $b \leq 2$

	$\mathcal{C}_7$				$\mathcal{C}_8$				$\mathcal{C}_9$			
(a,b) class	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS	Total LETS	Total EAS	Total FEAS
(2,2)	96	0	0	0	176	0	0	0	384	0	0	0
(3,2)	72	0	0	0	132	0	0	0	288	0	0	0
(4,2)	192	144	144	0	176	88	88	0	192	0	0	0
(5,2)	720	216	216	0	616	308	308	0	384	288	288	0
(6,1)	48	48	48	0	44	44	44	0	96	96	96	0
(6,2)	2556	1068	1068	0	1958	418	418	0	1200	144	144	0
(7,1)	336	240	240	0	176	88	88	0	192	0	0	0
(7,2)	9264	3600	3600	0	5192	1144	1144	0	2880	672	672	0
(8,0)	48	48	48	48	0	0	0	0	0	0	0	0
(8,1)	1176	720	720	0	440	220	220	0	288	96	96	0
(8,2)	35040	13464	13464	0	16104	5368	5368	0	10176	2304	2304	0
Search w. (6.1)	301 min.				115 min.				43 min.			
Search w. (6.2)	69 min.				31 min.				13 min.			
	$\mathcal{C}_{10}$				$\mathcal{C}_{11}$				$\mathcal{C}_{12}$			
(2,2)	144	0	0	0	264	0	0	0	576	0	0	0
(3,2)	192	72	0	0	264	44	0	0	576	96	0	0
(4,2)	312	0	0	0	308	0	0	0	672	0	0	0
(5,2)	408	24	0	0	396	44	0	0	864	96	0	0
(6,2)	648	144	72	0	572	88	44	0	1152	96	96	0
(7,2)	1272	216	216	0	880	44	44	0	1536	0	0	0
(8,1)	48	48	48	0	44	44	44	0	96	96	96	0
(8,2)	2952	912	768	0	1584	308	176	0	2304	384	192	0
Search w. (6.1)	17 min.				8 min.				4 min.			
Search w. (6.2)	5 min.				2 min.				2 min.			

Remark 9. *It is interesting to note that as mentioned in Subsection 6.4.2, less than 1 % of the run-time reported for each code is for finding ETSLs. For example, while most of the ETSs of C_{15}, reported in Table 6.10 are ETSLs, it takes the Algorithm 5 only 2 seconds to find all the ETSLs of the code.*

In Table 6.11, we have also reported the multiplicity of instances of ETSs and TSs obtained by the non-exhaustive search algorithms of [47] and [1], respectively. As can be seen, there are some cases in Table 6.11, where the multiplicity of ETS classes obtained here differs from the multiplicity of ETS classes reported in [47] and TS classes reported in [1]. These cases are boldfaced in the table.

In [47] and [53], for irregular codes, the authors relaxed the condition that degree-2

Table 6.10: Multiplicities of (a, b) ETSs, LETSs, EASs and FEASs of Codes $\mathcal{C}_{13}$, $\mathcal{C}_{14}$ and $\mathcal{C}_{15}$ within the range of $a \leq 10$ and $b \leq 2$

	$\mathcal{C}_{13}$					$\mathcal{C}_{14}$					$\mathcal{C}_{15}$			
(a,b) class	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS [23]	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS [23]	Total ETS	Total LETS	Total EAS	Total FEAS
(2,2)	240	0	0	0	240	440	0	0	0	440	960	0	0	0
(3,2)	216	0	0	0	216	396	0	0	0	396	864	0	0	0
(4,2)	192	0	0	0	192	352	0	0	0	352	768	0	0	0
(5,2)	168	0	0	0	168	308	0	0	0	308	672	0	0	0
(6,2)	216	72	72	0	216	352	88	88	0	352	672	96	96	0
(7,2)	408	24	24	0	408	572	0	0	0	572	768	0	0	0
(8,2)	624	24	24	0	624	792	0	0	0	792	864	0	0	0
(9,2)	912	120	120	0	912	968	44	44	0	968	1152	192	192	0
(10,1)	0	0	0	0	0	44	44	44	0	44	0	0	0	0
(10,2)	1560	168	168	0	1560	1276	88	88	0	1276	1728	0	0	0
Search w. (6.1)	38 min.					18 min.					9 min.			
Search w. (6.2)	12 min.					7 min.					3 min.			

Table 6.11: Multiplicities of (a, b) ETSs, LETSs, EASs and FEASs of Code $\mathcal{C}_{16}$ within the range of $a \leq 12$ and $b \leq 2$

	$\mathcal{C}_{16}$					
(a,b) class	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS [47]	Total TS [1]
(2,2)	810	0	0	0	810	nr
(3,2)	729	0	0	0	729	nr
(4,2)	648	0	0	0	648	648
(5,2)	567	0	0	0	567	567
(6,2)	486	0	0	0	486	486
(7,2)	486	81	81	0	486	**485**
(8,2)	648	81	81	0	648	**637**
(9,2)	972	0	0	0	972	nr
(10,2)	1377	81	81	0	1377	**1210**
(11,2)	2106	324	324	0	**1944**	**1635**
(12,1)	81	81	81	0	81	81
(12,2)	3564	324	324	0	**2754**	**2166**
Search w. (6.1)	2 min.					
Search w. (6.2)	1 min.					

Table 6.12: Multiplicities of (a, b) ETSs, LETSs, EASs and FEASs of Codes $\mathcal{C}_{17}$ and $\mathcal{C}_{18}$ within the range of $a \leq 10$ and $b \leq 2$

	$\mathcal{C}_{17}$								$\mathcal{C}_{18}$					
(a, b) class	Total ETS	Total LETS	Total EAS	Total FEAS	Total ETS [23]	Total FEAS*	Total FEAS [47]	Total FAS [53]	Total ETS	Total LETS	Total EAS	Total FEAS	Total FEAS*	Total FAS [53]
(2,2)	230	0	0	0	230	230	nr	230	916	0	0	0	458	458
(3,2)	219	0	0	0	219	219	219	219	439	0	0	0	439	439
(4,2)	208	0	0	0	208	208	208	208	420	0	0	0	420	420
(5,2)	198	0	0	0	198	198	198	198	404	0	0	0	404	404
(6,2)	207	19	0	0	207	205	205	205	388	0	0	0	387	387
(7,1)	2	2	2	0	2	2	2	2	1	1	1	0	1	1
(7,2)	276	24	24	0	276	271	271	271	406	30	30	0	403	403
(8,1)	8	4	4	0	8	8	8	8	4	2	2	0	4	4
(8,2)	466	61	60	0	466	458	458	458	524	45	44	0	519	519
(9,1)	16	4	4	0	16	16	16	16	8	2	2	0	8	8
(9,2)	870	75	74	0	870	855	855	nr	806	52	50	0	795	nr
(10,1)	22	3	3	0	22	22	22	22	14	4	4	0	14	14
(10,2)	1640	168	167	0	1640	1593	**1533**	nr	1305	73	73	0	1290	nr
Search w. (6.1)	20 min.								21 min.					
Search w. (6.2)	11 min.								11 min.					

variable nodes of (fully) absorbing sets must be connected to two satisfied check nodes. To compare our results for Codes C_{17} and C_{18} with those reported in [47] and [53], we have also reported the list of FEASs of these two codes with this modification in the definition of absorbing sets and fully absorbing sets. These results are identified with a star in Table 6.12. As can be seen, for both codes, these results match those obtained by the exhaustive search algorithm of [53], in all the classes where the results are reported in [53]. Comparison with the non-exhaustive results of [47], however, shows a discrepancy in the multiplicity for the $(10, 2)$ class.

We perform Monte Carlo simulations on codes $\mathcal{C}_4$ and $\mathcal{C}_7$ with a 3-bit quantized min-sum decoder over the AWGN channel to obtain 100 block errors at different SNR. Fig. 6.7 shows the Monte Carlo simulations for code $\mathcal{C}_4$. For this code, at SNR 5 dB (which is in the error floor region of this code), among the 100 cases, 99 of them were ETSs and 91 of them were ETSs within the interest range of Table 6.7. Also, 89 error-prone structures of these 91 ones are LETSs. The error patterns in which the decoder was trapped were LETSs that are listed in Table 6.7: $23 \times (7, 2)$, $17 \times (6, 2)$, $14 \times (9, 1)$, $13 \times (8, 2)$, $8 \times (5, 3)$, $6 \times (9, 3)$, $5 \times (8, 3)$ and the other 3 patterns. Fig.

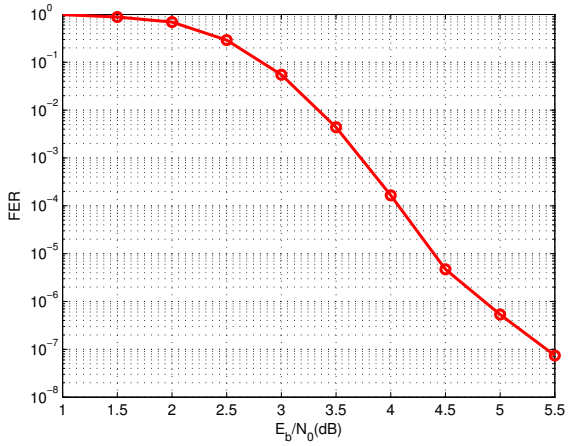


Figure 6.7: Simulation results of $\mathcal{C}_4$, decoded by the 3-bit min-sum decoder.

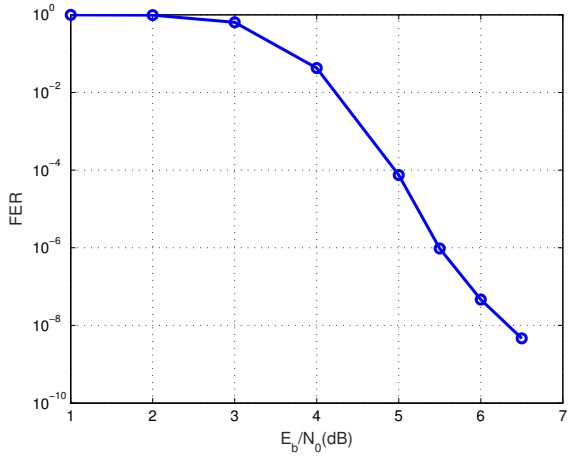


Figure 6.8: Simulation results of $\mathcal{C}_7$, decoded by the 3-bit min-sum decoder.

6.8 shows the Monte Carlo simulations for code $\mathcal{C}_7$. Also for this code, at SNR 6.5 dB (which is in the error floor region of this code), among all 100 cases, 96 of them were ETSs and 72 of them were ETSs within the interest range of Table 6.7. Also, 66 error-prone structures of these 72 ones are LETSs and 55 of them are LETSs in the $(7, 1)$ class.

Chapter 7

Lower Bounds on the Size of Smallest Elementary and Non-Elementary Trapping Sets in Variable-Regular LDPC Codes

7.1 Introduction

In this chapter, we derive a lower bound on the size of the smallest possible elementary and non-elementary trapping sets with a given b in variable-regular LDPC codes. While reported experimental results show that the vast majority of harmful trapping sets are elementary, to the best of our knowledge, no theoretical result is available to justify such results. It is, however, well-known that smaller trapping sets are often more harmful. This is due to the fact that, in the error-floor region, where the probability of channel errors is very low, the chances of the decoder being trapped into a TS decreases rapidly with the increase in the size of the TS. We derive a lower bound on the size of the smallest possible non-elementary trapping sets (NETS) with a given b in variable-regular LDPC codes, and demonstrate that this lower bound is larger (in some cases, significantly) compared to the smallest size of ETSs with the same b value. This provides a theoretical justification for why NETSs are, in general, less harmful than their ETS counterparts.

$$a \geq \begin{cases} k + T\sum_{i=0}^{\lfloor g/4-2 \rfloor}(d_{\mathrm{v}}-1)^i, & \text{for } g/2 \text{ even}, \\ k + T\sum_{i=0}^{\lfloor g/4-2 \rfloor}(d_{\mathrm{v}}-1)^i + max\{\lceil (T(d_{\mathrm{v}}-1)^{\lfloor g/4-1 \rfloor})/d_{\mathrm{v}} \rceil, (d_{\mathrm{v}}-1-\lfloor b'/k \rfloor)(d_{\mathrm{v}}-1)^{\lfloor g/4-1 \rfloor}\}, & \text{for } g/2 \text{ odd}, \end{cases} \tag{7.1}$$

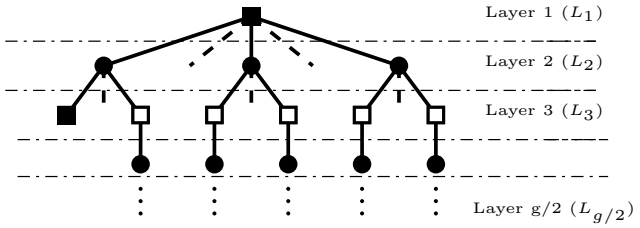


Figure 7.1: A tree-like expansion of a TS rooted at a check node of degree k.

7.2 Lower bounds on the size of smallest TSs

The presence of a degree-k check node in the induced subgraph $G(\mathcal{S})$ of a trapping set $\mathcal{S}$ of a variable-regular Tanner graph G is studied in the following theorem.

Theorem 5. *Consider a variable-regular Tanner graph G with variable degree d_{v} and girth g. A lower bound on the size a of an (a, b) trapping set in G, whose induced subgraph contains a check node of degree k (≥ 2) is given in (9.1), where $b' = b - (k \mod 2)$, $T = k(d_{\mathrm{v}}-1) - b'$, and b is assumed to satisfy $b < k(d_{\mathrm{v}}-1) + (k \mod 2)$.*

Proof. To prove the result, we consider the tree-like expansion of the induced subgraph of the TS starting from the degree-k check node c as the root with $g/2$ layers (depth $g/2-1$). Fig. 9.1 shows such an expansion. Since we are interested in a TS with minimum size, we consider a case where the degree of all the other check nodes in the subgraph is either 2 or 1, depending on whether they are satisfied or unsatisfied, respectively. In this tree, node c, in layer one (L_1), is connected to k variable nodes, in layer two (L_2), and each variable node in L_2 is connected to $d_{\mathrm{v}}-1$ check nodes in L_3. To minimize the size of the TS, we assume that all the remaining b' unsatisfied check nodes are in this layer. (If k is odd, node c itself is unsatisfied, and $b' = b-1$. Otherwise, $b' = b$.) Therefore, there are T degree-2 check nodes, and T degree-d_{v} variable nodes in L_3 and L_4, respectively. For the rest of the variable layers, number of variable nodes at layer L_{2i+2} is $d_{\mathrm{v}}-1$ times the number of variable nodes at layer L_{2i}, for $i = 2, 3, \ldots, \lfloor g/4-1 \rfloor$.

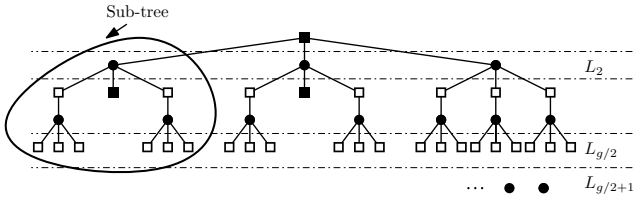


Figure 7.2: A tree with 5 layers rooted at a degree-3 check node within the induced subgraph of a NETS in a variable-regular graph with $d_{\rm v} = 4$ and $g = 10$.

To satisfy the constraint of having a Tanner graph with girth g, all the variable and check nodes in the first $g/2$ layers of the tree should be distinct. Based on the construction of the tree, it is easy to see that at layer $L_{2i+2}, 1 \leq i \leq \lfloor g/4 - 1 \rfloor$, of the tree, we have $T(d_{\rm v}-1)^{i-1}$ variable nodes, for a total of $k + \sum_{i=0}^{\lfloor g/4-2 \rfloor} T(d_{\rm v}-1)^i$, in the first $g/2$ layers.

For even values of $g/2$, the tree will end at variable nodes in the last layer, $L_{g/2}$, and thus the lower bound of (9.1) follows.

For odd values of $g/2$, the tree will end at check nodes in layer $L_{g/2}$. So far, there are $k + \sum_{i=0}^{\lfloor g/4-2 \rfloor} T(d_{\rm v}-1)^i$ distinct variable nodes in the tree (subgraph), and $(T(d_{\rm v}-1)^{\lfloor g/4-1 \rfloor})$ distinct check nodes in the last layer, $L_{g/2}$. To complete the subgraph of the TS, at least $\mathcal{L}_1 = \lceil (T(d_{\rm v}-1)^{\lfloor g/4-1 \rfloor})/d_{\rm v} \rceil$ variable nodes are needed in layer $L_{g/2+1}$ to be connected to the check nodes in layer $L_{g/2}$. On the other hand, in the tree, there are k sub-trees, each starting from one variable node at L_2. One can easily see that to avoid having cycles shorter than g in the subgraph, any new variable node at $L_{g/2+1}$ can only be connected to the check nodes of each such sub-tree at most once (see, Fig. 7.2). Therefore, at $L_{g/2+1}$, we need, at least, as many variable nodes as check nodes of each sub-tree at $L_{g/2}$. To derive the lower bound, we need to minimize the maximum number of such check nodes in each sub-tree. Therefore, all the b' unsatisfied check nodes at L_3 should be allocated to the k sub-trees as equally as possible (either exactly in equal numbers if b' is divisible by k, or with the difference of at most one between the subtrees, otherwise). This results in the maximum of $\mathcal{L}_2 = (d_{\rm v} - 1 - \lfloor b'/k \rfloor)(d_{\rm v}-1)^{\lfloor g/4-1 \rfloor})$ check nodes in each subtree at $L_{g/2}$, which is also equal to a lower bound on the minimum number of variable nodes needed at $L_{g/2+1}$. Therefore, at least $max\{\mathcal{L}_1, \mathcal{L}_2\}$ variable nodes are needed to complete the subgraph. ■

Note that if $b = k(d_v - 1) + (k \mod 2)$, then $T = 0$, and the lower bounds provided in (9.1) are both equal to k and still valid.

Remark 10. *Based on Lemma 1, the bound provided in (9.1) can sometimes be improved by one. For example, based on Theorem 5, the size of smallest possible* $(a, 2)$ *TS containing a degree-*3 *check node, in variable-regular graphs with* $d_v = 3$ *and* $g = 6$ *is* 5*. However, based on Lemma 1, no* $(5, 2)$ *TS can exist in a variable-regular graph with* $d_v = 3$*. The lower bound in this case is thus improved by one to* 6*.*

Corollary 3. *Theorem 5 with* $k = 2$ *provides a lower bound on the size of the smallest possible ETSs.*

We note that the bound of Corollary 3 applies to both LETSs and ETSLs. This bound however, can often be improved, for each category of LETSs and ETSLs, based on the results provided in Chapters 5 and 6, respectively. LETSs for variable-regular LDPC codes were fully characterized in Chapter 5. One can thus use the characterization algorithm to obtain the size of the smallest (a, b) LETS structure for any value of b in a variable-regular Tanner graph with a given variable-degree d_v and girth g. ETSLs for irregular LDPC codes, on the other hand, were fully characterized in Chapter 6. ETSLs were partitioned into two categories: (i) ETSLs that contain at least one LETS sub-structure, denoted by ETSL_1, and (ii) those that do not contain any LETS sub-structure, denoted by ETSL_2. ETSL_1 structures are obtained from their largest LETS substructure by successively appending single variable nodes, through a single connection, to the unsatisfied check nodes of the LETS structure and its subsequent children. We thus have the following results for ETSLs in variable-regular LDPC codes.

Proposition 11. *If for a given* b $(b > d_v - 2)$*, the size of the smallest* ETSL_1 *in variable-regular graphs with variable degree* d_v *is* a^**, then there must exist either a LETS structure or an* ETSL_1 *structure in the* $(a^* - 1, b' = b - d_v + 2)$ *class. In either case, the value* $a^* - 1$ *is the size of the smallest TS (in the corresponding category of LETS or* ETSL_1*) in all the classes with the number of unsatisfied check nodes equal to* b'*.*

Proposition 11 allows us to find the minimum size of (a, b) ETSL_1 structures for a given value of b, using similar results obtained for LETS structures based on the characterization algorithm of Chapter 5. We also note that since any ETSL_1 structure

has at least one leaf (in its normal graph), it must have at least $d_v - 1$ unsatisfied check nodes, thus the following remark.

Remark 11. *In variable-regular graphs with variable degree d_v, there does not exist any (a, b) ETSL$_1$ structure with $b < d_v - 1$.*

It was shown in Chapter 6 that ETSL$_2$ structures contain no cycles, and are thus trees. All ETSL$_2$ structures for variable-regular graphs are thus in $(a, a(d_v - 2) + 2)$ classes, depending on their size a. The following result is then easy to prove.

Proposition 12. *In a variable-regular graph with variable degree d_v, there does not exist any (a, b) ETSL$_2$ structure with $b < d_v$. For $b \geq d_v$, the size of ETSL$_2$ structures is limited to only $a = (b - 2)/(d_v - 2)$ (this is only if a is a positive integer and the conditions of Lemma 1 are satisfied).*

NETS structures have at least one check node with degree 3 or larger in their subgraph. We thus have the following corollary of Theorem 5.

Corollary 4. *Theorem 5 with $k = 3$ ($k = 4$) provides a lower bound on the size of the smallest possible NETSs with $b > 0$ ($b = 0$).*

The results for the minimum sizes of LETS and ETSL structures along with the lower bound of Corollary 3 on the size of ETSs, and the lower bound of Corollary 4 on the size of NETSs are reported in Table 7.1. The results are given for variable-regular graphs with $d_v = 3, 4, 5, 6$, and girths 6, 8, 10, and for values of $b = 0$ to 5.

In Table 7.1, having the symbol "-" for ETSLs means that it is not possible to have an ETSL structure for the specific value of b in variable-regular graphs with the given g and d_v. Having the symbol "-" for ETSs (that appears in the entries corresponding to $d_v = 3$, and $b = 4, 5$) is stemming from the upper bound on the values of b in Theorem 5. Moreover, the symbol "?" in the table indicates that finding the smallest size LETS or ETSL was too complex to be performed in a day on a desktop computer with 2.4-GHz CPU and 8-GB RAM.

It should be noted that the lower bounds of Corollary 4 on the minimum size of NETSs are tight in many cases reported in Table 7.1. Figure 7.3 shows three examples of NETS structures in variable-regular graphs where the lower bound is tight. Moreover, one can see that except for few cases, in all the cases reported in Table 7.1, the lower bounds on the smallest size of NETSs are larger, in some cases

Table 7.1: The smallest size of ETSs (LETSs and ETSLs), and lower bounds of Corollaries 3 and 4 on the size of ETSs and NETSs, respectively, for Tanner graphs with $d_v = 3, 4, 5, 6$, $g = 6, 8, 10$, and for $b \leq 5$

$NETS$ / $ETS^{(LETS)}_{(ETSL)}$		$d_v = 3$			$d_v = 4$			$d_v = 5$			$d_v = 6$		
		$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$
	$b = 0$	8 – – – $4^{(4)}_{(-)}$	12 – – – $6^{(6)}_{(-)}$	18 – – – $10^{(10)}_{(-)}$	7 – – – $5^{(5)}_{(-)}$	16 – – – $8^{(8)}_{(-)}$	25 – – – $17^{(19)}_{(-)}$	8 – – – $6^{(6)}_{(-)}$	20 – – – $10^{(10)}_{(-)}$	36 – – – $26^{(30)}_{(-)}$	9 – – – $7^{(7)}_{(-)}$	24 – – – $12^{(12)}_{(-)}$	49 – – – $37^{(?)}_{(-)}$
	$b = 1$	5 – – – $5^{(5)}_{(-)}$	9 – – – $5^{(7)}_{(-)}$	13 – – – $9^{(11)}_{(-)}$	no TS			7 – – – $7^{(7)}_{(-)}$	15 – – – $9^{(13)}_{(-)}$	31 – – – $25^{(?)}_{(-)}$	no TS		
	$b = 2$	6 – – – $4^{(4)}_{(6)}$	8 – – – $4^{(6)}_{(8)}$	12 – – – $6^{(10)}_{(12)}$	6 – – – $4^{(5)}_{(-)}$	11 – – – $7^{(8)}_{(-)}$	20 – – – $12^{(19)}_{(-)}$	8 – – – $5^{(6)}_{(-)}$	14 – – – $8^{(10)}_{(-)}$	30 – – – $20^{(?)}_{(-)}$	8 – – – $7^{(7)}_{(-)}$	17 – – – $10^{(12)}_{(-)}$	42 – – – $30^{(?)}_{(-)}$
	$b = 3$	5 – – – $3^{(3)}_{(5)}$	7 – – – $3^{(5)}_{(7)}$	11 – – – $5^{(9)}_{(11)}$	no TS			7 – – – $5^{(7)}_{(-)}$	13 – – – $7^{(11)}_{(-)}$	29 – – – $19^{(?)}_{(-)}$	no TS		
	$b = 4$	4 – – – $-^{(4)}_{(4)}$	6 – – – $-^{(4)}_{(6)}$	8 – – – $-^{(8)}_{(10)}$	5 – – – $3^{(4)}_{(6)}$	9 – – – $4^{(7)}_{(9)}$	15 – – – $7^{(18)}_{(20)}$	6 – – – $4^{(6)}_{(8)}$	12 – – – $6^{(10)}_{(14)}$	24 – – – $14^{(?)}_{(?)}$	7 – – – $5^{(7)}_{(-)}$	15 – – – $8^{(12)}_{(-)}$	35 – – – $23^{(?)}_{(-)}$
	$b = 5$	5 – – – $-^{(5)}_{(5)}$	5 – – – $-^{(5)}_{(5)}$	7 – – – $-^{(5)}_{(9)}$	no TS			7 – – – $5^{(5)}_{(7)}$	11 – – – $5^{(9)}_{(11)}$	23 – – – $13^{(?)}_{(?)}$	no TS		

significantly, than those of ETSs. For example, in variable-regular graphs with $d_v = 3$ and $g = 10$, the smallest size of ETSs and NETSs with $b = 1$ are respectively 9 and 13. Figure 7.4 shows examples of LETS and NETS structures in the $(5, 1)$ and $(7, 1)$ classes of variable-regular graphs with $d_v = 3, g = 6$ and $d_v = 5, g = 6$, respectively. These are related to the only two cases in Table 7.1, where the minimum sizes of NETSs and LETSs are equal for values of $b \leq 2$. The structures in Figs. 7.4(a)-(b) are the only non-isomorphic LETS and NETS in the $(5, 1)$ class.

Example 23. *Using the characterization algorithm of Chapter 5, one can find all the non-isomorphic LETS structures of variable-regular graphs with $d_v = 3, g = 8$, in the range of $a \leq 10$ and $b \leq 2$. There are 48 such non-isomorphic structures. The only ETSL structures in the same range are in $(8, 2)$ and $(10, 2)$ classes, with only 1 and 4 non-isomorphic structures in each class, respectively. These correspond to 1 and 4 non-isomorphic LETS structures in the $(7, 1)$ and $(9, 1)$ classes, respectively. Moreover, based on Table 7.1, the only possible NETS structures in the interest range*

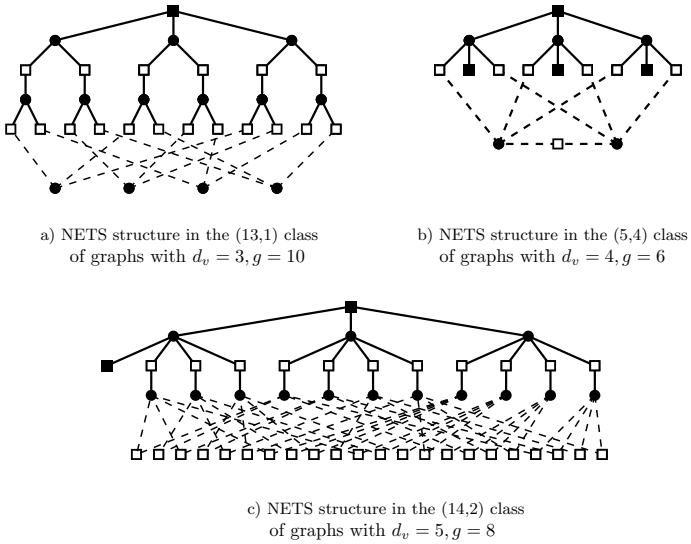


Figure 7.3: Examples of smallest NETS structures in graphs with different variable degrees and girths.

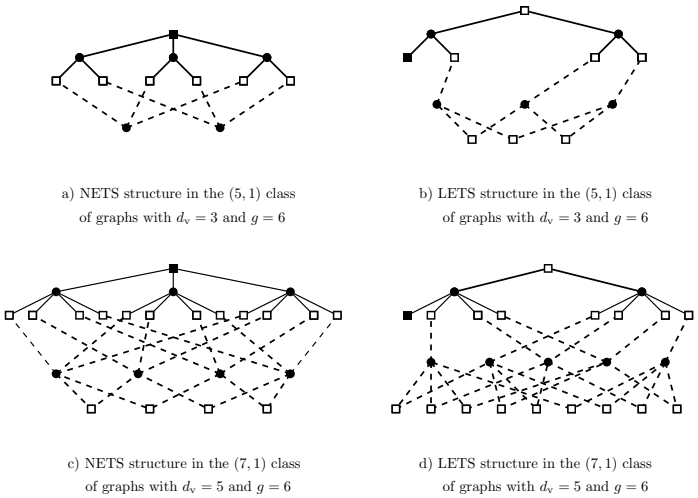


Figure 7.4: LETS and NETS structures in the $(5, 1)$ and $(7, 1)$ classes of variable-regular graphs with $d_{\mathrm{v}} = 3, g = 6$ and $d_{\mathrm{v}} = 5, g = 6$, respectively.

are in $(8, 2)$, $(9, 1)$ *and* $(10, 2)$ *classes. Table 7.2 shows the possible classes of LETS, ETSL and NETSs with* $a \leq 10$ *and* $b \leq 2$ *in variable-regular graphs with* $d_v = 3$ *and* $g = 8$*. The multiplicity of non-isomorphic structures for LETSs and ETSLs is given in parentheses. Table 7.2 clearly shows that for each value of b, the minimum size of LETS structures is smaller than that of NETS structures.*

Table 7.2: Trapping sets of Variable-Regular Graphs with $d_v = 3$ and $g = 8$ for $a \leq 10$ and $b \leq 2$

	$a = 4$	$a = 5$	$a = 6$	$a = 7$	$a = 8$	$a = 9$	$a = 10$
b = 0	-	-	$LETS(1)$	-	$LETS(2)$	-	$LETS(6)$
b = 1	-	-	-	$LETS(1)$	-	$LETS(4)$ $NETS$	-
b = 2	-	-	$LETS(1)$	-	$LETS(5)$ $ETSL(1)$ $NETS$	-	$LETS(28)$ $ETSL(4)$ $NETS$

Tanner (155,64) code [82] is an example of a variable-regular code with $d_v = 3$ *and* $g = 8$*. Within the range* $a \leq 10$ *and* $b \leq 2$*, this code has* 465 *and* 1395 *instances of LETSs in* $(8, 2)$ *and* $(10, 2)$ *classes, respectively (Chapter 5). These structures are known to be the dominant TSs of this code [46]. The same TSs are also identified in [97] (using an exhaustive search) to be the only TSs of this code in this range, which implies that the code does not have any NETSs in the range* $a \leq 10$ *and* $b \leq 2$*.*

Chapter 8

Tight Lower and Upper Bounds on the Minimum Distance of LDPC Codes

8.1 Introduction

In this chapter, we use the graphical structure of non-zero codewords within the Tanner graph of an LDPC code to devise our search algorithm, and to derive bounds on the minimum distance. In Section 8.2, we derive the lower bound L_{ne} for regular and irregular LDPC codes as a function of girth and the variable node degree distribution. In Section 8.3, we explain the modifications made to the algorithms of Chapters 5 and 6 to increase their reach for finding higher weight elementary codewords. This is used to derive an upper bound on d_{min}. Finally, numerical results are provided in Section 10.4.

8.2 Lower bound on the minimum distance of LDPC codes

The minimum distance of an LDPC code, like that of any other linear block code, is characterized by the minimum weight of non-zero codewords. A non-zero codeword of weight a in an LDPC code can be viewed as an $(a, 0)$ trapping set (TS) in the code's Tanner graph, where the non-zero elements of the codeword correspond to the variable nodes of the TS. In this context, the graphical representation of a codeword is the subgraph of the corresponding TS. In this work, we partition the non-zero codewords of an LDPC code into two categories of *elementary* and *non-elementary*, where in

the first category, all the satisfied check nodes have degree 2. The following lemmas provide graphical characterizations of elementary and non-elementary codewords.

Lemma 9. *Consider an LDPC code with minimum variable degree at least two. Any elementary codeword of such an LDPC code corresponds to a leafless elementary trapping set (LETS) in the code's Tanner graph.*

Proof. By definition, an elementary codeword corresponds to an ETS. Now assume that such an ETS is not leafless. This implies that there exists a variable node v in the ETS that is connected only to one satisfied check node of degree 2. This together with the fact that variable nodes have degree at least two, implies that v is connected to at least one unsatisfied check node, which is in contradiction with the ETS representing a codeword. ■

Lemma 10. *Consider an LDPC code with minimum variable degree at least two. Any non-elementary codeword of such an LDPC code corresponds to a leafless trapping set in the code's Tanner graph with at least one check node of degree at least 4, where the term "leafless" here means that every variable node is connected to at least two satisfied check nodes.*

Proof. The proof of the TS having to be leafless is similar to that presented in the proof of Lemma 9. The fact that the codeword is not elementary implies that there must be at least one satisfied check node of degree larger than 2 in the TS. ■

In the following, we establish a lower bound on the size of non-zero codewords of an LDPC code using the graphical representation of the corresponding TS. The results for variable-regular codes are presented in Subsection 8.2.1, followed by those for irregular codes in Subsection 8.2.2.

8.2.1 Variable-Regular LDPC Codes

Based on Theorem 5, the following theorem establishes a lower bound on the weight of a non-zero codeword of a variable-regular LDPC code, where the corresponding TS has at least one even-degree check node of degree k.

Theorem 6. *In a variable-regular Tanner graph with variable degree d_v and girth g, the weight a of any non-zero codeword, whose subgraph contains at least one degree-k*

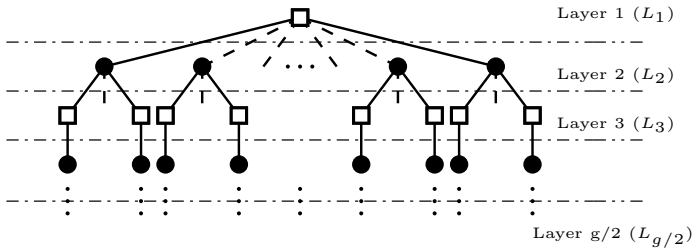


Figure 8.1: A tree within the induced subgraph of a minimum weight codeword rooted at a check node of degree k.

$(k \geq 2)$ check node, satisfies

$$a \geq \begin{cases} \sum_{i=0}^{\lfloor g/4-1 \rfloor} k(d_v-1)^i + max\{\lceil (k(d_v-1)^{\lfloor g/4 \rfloor})/d_v \rceil, (d_v-1)^{\lfloor g/4 \rfloor}\}, & \text{for } g/2 \text{ odd}, \\ \sum_{i=0}^{\lfloor g/4-1 \rfloor} k(d_v-1)^i, & \text{for } g/2 \text{ even}. \end{cases} \tag{8.1}$$

Remark 12. *Based on Lemma 1, for odd values of d_v, there is no TS in the $(a, 0)$ class with an odd value of a. In such cases, therefore, the lower bound of Theorem 6 can be improved, if the value in (8.1) happens to be an odd number.*

Example 24. *For the case of variable-regular graphs with $d_v = 3$ and $g = 6$, based on Theorem 6, the weight of a codeword whose subgraph contains at least one degree-4 check node $(k = 4)$ is lower bounded by 7. However, based on Lemma 1, for an odd value of d_v, there is no trapping set in any $(a, 0)$ class with odd value of a. Therefore, the above mentioned lower bound is increased to 8 from 7.*

Corollary 5. *The result of Theorem 6 with $k = 2$ provides a lower bound on d_{min} for variable-regular LDPC codes.*

Proof. It is easy to see that (8.1) is an increasing function of k. Therefore, the lower bound on the smallest weight of a codeword is obtained by setting $k = 2$ in (8.1). This corresponds to an elementary codeword. ■

We note that the result of Corollary 19 is essentially the same as the lower bound obtained in [82], [89], on the minimum distance of variable-regular LDPC codes.

The TS corresponding to a non-elementary codeword has at least one check node with degree 4 or larger. We thus have the following corollary of Theorem 6, which establishes a lower bound on the minimum weight of non-elementary codewords.

Table 8.1: Lower bounds on the minimum weight of non-elementary (elementary) codewords of variable-regular graphs with different d_v and g

	$d_v = 2$	$d_v = 3$	$d_v = 4$	$d_v = 5$
$g = 6$	6 (3)	8[1] (4)	7 (5)	8 (6)
$g = 8$	8 (4)	12 (6)	16 (8)	20 (10)
$g = 10$	10 (5)	18 (10)	25 (17)	36 (26)
$g = 12$	12 (6)	28 (14)	52 (26)	84 (42)

Corollary 6. *The result of Theorem 6 with $k = 4$ provides a lower bound on the minimum weight of non-elementary codewords in variable-regular LDPC codes.*

Table 8.1 shows the lower bounds on the minimum weight of elementary and non-elementary codewords of variable-regular LDPC codes with $d_v = 2, 3, 4, 5$, and $g = 6, 8, 10, 12$, obtained through Corollaries 19 and 6, respectively (the numbers in the parentheses correspond to the elementary codewords). Through Table 8.1, one can see the difference between the smallest possible weight of elementary codewords and that of non-elementary codewords. The difference is particularly large for codes with larger girth and larger variable degree.

Remark 13. *It should be noted that the lower bounds of Corollaries 19 and 6 are tight in many cases. In particular, all the lower bounds on the lowest weight of elementary codewords listed in Table 8.1 are tight, except the cases of $(d_v = 4, g = 10)$ and $(d_v = 5, g = 10)$. In these cases, the smallest possible weight of an elementary codeword is* 19 *and* 30*, respectively.*[2]

Fig. 8.2 shows four examples of lowest weight non-elementary codewords of LDPC codes with different variable degrees and girths. One can see that for all four examples, the codeword weight is the same as the lower bound provided in Table 8.1 for non-elementary codewords. (In Fig. 8.2, Solid and dashed lines are used to represent the connections of variable and check nodes in layers $L_1, \ldots, L_{g/2}$, and in layers beyond $L_{g/2}$, respectively.)

To improve the lower bound of Corollary 19, which is the same as those in [82], [89], we use the fact that elementary codewords, as a special case of LETSs, have a graphical structure that lends itself well to the *dpl* exhaustive search algorithm of Chapter

[1] The entry for elementary codewords in this case has been improved from 7 to 8, based on Remark 12.

[2] These values are obtained using the *nauty* program [60].

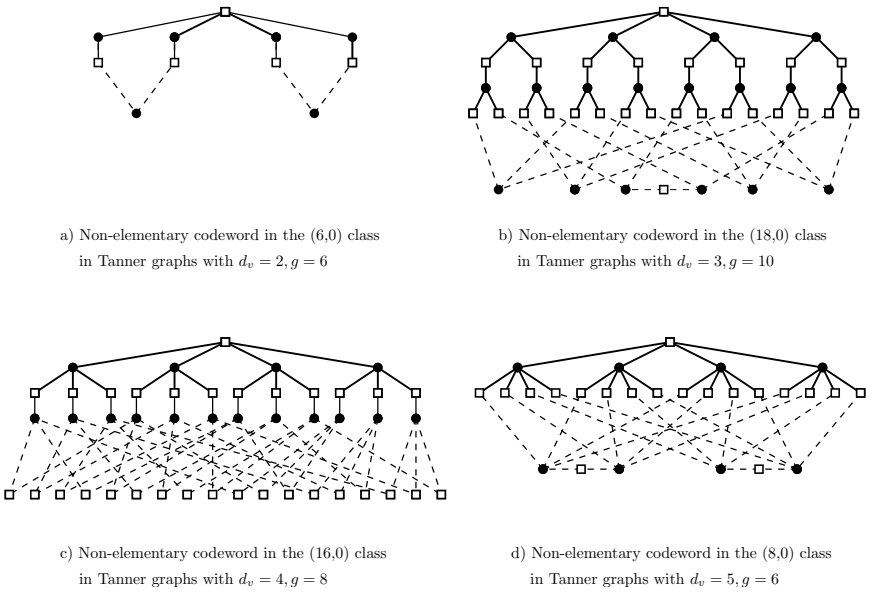


Figure 8.2: Examples of lowest weight non-elementary codewords in variable-regular graphs with (a) $d_{\mathrm{v}} = 2, g = 6$, (b) $d_{\mathrm{v}} = 3, g = 10$, (c) $d_{\mathrm{v}} = 4, g = 8$, and (d) $d_{\mathrm{v}} = 5, g = 6$.

5. Using the *dpl* search algorithm with $b_{max} = 0$, we can efficiently and exhaustively find all the elementary codewords of a variable-regular Tanner graph with weight at most a_{max}. We set a_{max} to be equal to the lower bound of Corollary 6 (on the lowest weight of non-elementary codewords), which we denote by L_{ne}, hereafter. If the search algorithm finds at least one elementary codeword in this range, we know that the smallest weight of the found codewords is the exact minimum distance of the code. Otherwise, we are able to establish the lower bound of L_{ne} on d_{min}, i.e., $d_{min} \geq L_{ne}$.

In our case, where $b_{max} = 0$, for given values of d_{v}, g, and $a_{max} = L_{ne}$, the characterization table includes the list of all the simple cycles that need to be enumerated to initiate the *dpl* search process, plus all the different expansion techniques that need to be applied to LETSs of different (a, b) classes. An example of the characterization table for the case of $d_{\mathrm{v}} = 3$, $g = 6$, and $a_{max} = L_{ne} = 8$, is provided in Table 8.2.

One can see that the lowest weight a in this row, which corresponds to the lowest weight of an elementary codeword, is 4. This matches the result of Corollary 19, as reflected in Table 8.1. The table also shows that there is only one structure for elementary codewords of weight 4. In the table, for each (a, b) class of LETSs, the

Table 8.2: Characterization of Elementary Codewords for Variable-Regular Graphs with $d_v = 3$ and $g = 6$ for $a \leq a_{max} = L_{ne} = 8$

	$a=3$	$a=4$	$a=5$	$a=6$	$a=7$	$a=8$
b = 0		1		2		5
b = 1			-		-	
b = 2		-		-		-
b = 3	$\boldsymbol{s_3}$ $---$ dot, lo_3^3		dot		dot	
b = 4		$\boldsymbol{s_4}$ $---$ dot, pa_2		dot		-

expansion techniques applied to all the LETS structures within the class is also shown. Empty entries in the table, obtained based on Lemma 1, show that it is impossible to have LETS structures in those classes. Having the symbol "-" as the only entry in a class means that no expansion technique is applied to the structures in that class.

8.2.2 Irregular LDPC Codes

The following result is the irregular counterpart of Theorem 6 for variable-regular LDPC codes, given in Subsection 8.2.1.

Theorem 7. *In an irregular Tanner graph with variable degree distribution $\lambda(x)$ and girth g, the weight a of any non-zero codeword, whose subgraph contains at least one degree-k ($k \geq 2$) check node, satisfies*

$$a \geq \begin{cases} \sum_{i=0}^{\lfloor g/4-1 \rfloor} k(d_{v_{min}}-1)^i + max\{\lceil (k(d_{v_{min}}-1)^{\lfloor g/4 \rfloor})/d_{v_{max}} \rceil, (d_{v_{min}}-1)^{\lfloor g/4 \rfloor}\}, & \text{for } g/2 \text{ odd}, \\ \sum_{i=0}^{\lfloor g/4-1 \rfloor} k(d_{v_{min}}-1)^i, & \text{for } g/2 \text{ even}, \end{cases} \tag{8.2}$$

where $d_{v_{min}}$ and $d_{v_{max}}$ are the minimum and maximum variable degrees in $\lambda(x)$, respectively.

Proof. The proof is similar to that of Theorem 6. In the tree subgraph of the codeword (up to layer $L_{g/2}$), we consider all the variable nodes to have the minimum degree $d_{v_{min}}$. This minimizes the number of check nodes in the following layers, ultimately minimizing the number of variable nodes in the subgraph. For the case where $g/2$ is

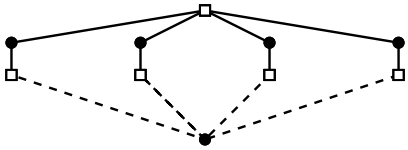


Figure 8.3: Lowest weight non-elementary codeword in the $(5, 0)$ class of irregular graphs with $g = 6$ that contain variable nodes with degrees 2 and 4.

an odd number, we have variable nodes in layer $L_{g/2+1}$ of the subgraph. To obtain a lower bound on the number of variable nodes in this layer, we assume the maximum variable node degree $d_{v_{max}}$ for the variable nodes in this layer. ■

For irregular LDPC codes, also, using $k = 2$ and $k = 4$ in Theorem 7 will result in lower bounds on the weight of elementary and non-elementary codewords, respectively.

Example 25. *Fig. 8.3 shows an example of a non-elementary codeword with lowest weight for irregular graphs with girth* 6 *containing variable nodes with degrees 2 and 4.*

We use an approach similar to the one described in Subsection 8.2.1 to either find the exact minimum distance of an irregular code, or to establish the lower bound of L_{ne} on d_{min} ($d_{min} \geq L_{ne}$), where L_{ne} is obtained by using $k = 4$ in (8.2). To search for elementary codewords of weight less than or equal to L_{ne}, we use the *dpl* search algorithm, devised in Chapter 6, for irregular LDPC codes.

8.3 Upper bound on the minimum distance of LDPC codes

If we fail to find the exact minimum distance of an LDPC code based on the approach described in Section 8.2, then, we have established that $d_{min} \geq L_{ne}$. For such cases, in this section, we also find an upper bound on d_{min}. To obtain this upper bound, we find an elementary codeword with weight larger than L_{ne}. This can be performed through the exhaustive dpl searches of Chapters 5 and 6, for regular and irregular graphs, respectively. The problem however, is that the complexity of such a search increases rather rapidly, if the range of search, indicated by the value of a_{max}, is increased much beyond the value of L_{ne}. This is explained in the following example.

Example 26. *For variable-regular LDPC codes with* $d_{\text{v}} = 3$ *and* $g = 8$*, we have* $L_{ne} = 12$*. We note that the number of non-isomorphic elementary codeword structures with weight a less than or equal to* $a_{max} = 8$*,* 12*,* 16*, and* 20*, is* 3*,* 31*,* 933*, and* 106284*, respectively.*[3] *One can see the rapid increase in the number of structures that one needs to search for as the range of search, i.e.,* a_{max}*, increases beyond* $L_{ne} = 12$*. Our experimental results show that while dpl exhaustive characterization/search algorithm can characterize/search all the possible elementary codewords for the first three cases, i.e.,* $a_{max} = 8$*,* 12*,* 16*, in a reasonable amount of time, the characterization/search in the fourth case, i.e.,* $a_{max} = 20$*, would take too long. To demonstrate how the runtime of the dpl search changes with the increase of the search range, we consider two LDPC codes with* $d_{\text{v}} = 3$ *and* $g = 8$*. The codes have block lengths and dimensions* $(n, k) = (155, 64)$ *[80] and* $(5219, 4300)$ *[81]. For the first code, on a desktop computer with* 2.4*-GHz CPU and* 8*-GB RAM, the dpl search takes* 2*,* 6*, and* 193 *seconds to search for all the elementary codewords of weight* $a \leq 8$*,* 12*, and* 16*, respectively. For the second code, the corresponding runtimes are* 1*,* 8*, and more than* 300 *minutes, respectively.*

To overcome the problem of high complexity of the exhaustive *dpl* search, in cases where the lowest weight of elementary codewords is well above L_{ne}, we modify the search such that it can handle larger values of a_{max}. This however, comes at the expense of losing the exhaustiveness of the search, and thus we are not guaranteed to find the elementary codewords with the lowest weight in our search.

In the original *dpl* characterization/search algorithm, to exhaustively cover all the (a, b) LETS structures in the interest range of $a \leq a_{max}$ and $b \leq b_{max}$, the algorithm sometimes needs to also cover auxiliary structures with their b values larger than b_{max}, and up to $b'_{max} > b_{max}$. (This is certainly the case for $b_{max} = 0$, which is what we are interested in.) For the exhaustive coverage in the interest range, the value of b'_{max}, obtained by *dpl* characterization, is proved to be minimal. In this part of the work, rather than selecting the b'_{max} as in the original *dpl* characterization/search, we choose it to be a smaller value. To compensate for the detrimental effect that this new choice will have on the exhaustiveness of the *dpl* search, rather than using the specific expansion techniques that the original *dpl* characterization determines for each LETS class, we apply *all* the possible expansions from the set of *dot*, *path* and *lollipop* expansions to LETS structures in each class in the range of $a \leq a_{max}$ and

[3] The non-isomorphic structures were enumerated using nauty program [60].

$b \leq b'_{max}$. The only constraint for the application of a certain expansion technique to LETS structures within a specific class is that the expanded structure must still remain within the range $a \leq a_{max}$ and $b \leq b'_{max}$. This is examined and imposed using the following proposition.

Proposition 13. *Suppose that $\mathcal{S}$ is an (a, b) LETS structure of variable-regular Tanner graphs with variable degree d_v. Then the expansion of $\mathcal{S}$ using dot_m, pa_m, and lo^c_m will result in LETS structure(s) in the $(a+1, b+d_v-2m)$, $(a+m, b-2+m(d_v-2))$, and $(a+m, b-2+m(d_v-2))$ classes, respectively.*

Example 27. *Consider the characterization table of the dpl search for elementary codewords of weight $a \leq 8$, in variable-regular graphs with $d_v = 3$ and $g = 6$, as presented in Table 8.2. In this case, $b_{max} = 0$, and $b'_{max} = 4 > b_{max}$. One can also see in the table the specific expansion techniques that are determined by the dpl characterization to be applied to different LETS classes. For example, the table shows that dot, pa_2 and dot expansions will need to be applied to LETS structures in $(4, 4)$ and $(5, 3)$ classes, respectively. Now, if we reduce the value of b'_{max} to 3 in the modified dpl search, we limit the algorithm to only search for and use LETS structures with $b \leq 3$. We however, allow any possible expansion that results in such LETS structures to be used in each class. This would mean, for example, that for the $(5, 3)$ class, we apply dot and pa_2 expansions rather than only dot expansion, which was used in the original dpl search.*

Given the values a_{max} and b'_{max}, Routine 13 provides a pseudo code for finding the list of expansion techniques that are required to be applied to all the non-isomorphic structures in each (a, b) LETS class of variable-regular and irregular graphs. These expansions are stored in the (a, b) entry of table $\mathcal{EX}$, $\mathcal{EX}_{(a,b)}$. The expansion *dot* is applied to all the (a, b) classes with $a \leq a_{max} - 1$. Also, pa_m and lo^c_m are applied to all the (a, b) classes with $a \leq a_{max} - m$. The only constraint for using an expansion technique is that the b value(s) of the new LETS structure(s) need to remain in the range identified by b'_{max}.

A pseudo code for obtaining an upper bound $d^{(u)}_{min}$ on d_{min} is presented in Algorithm 8. To start the algorithm, one can select a_{max} to be initially a rather large value a_0, say three or four times the value of L_{ne}, and choose b'_{max} to be a rather small value. This value for variable-regular codes is set at $b'_{max} = g/2(d_v - 2)$, in

Routine 13 (Expansions) Finds all the possible expansions $\mathcal{EX}_{(a,b)}$ for the (a, b) classes of LETS structures, $g/2 \leq a < a_{max}$, $1 \leq b \leq b'_{max}$, for a Tanner graph with girth g and variable degree $d_{\rm v}$ (for irregular graphs, $d_{\rm v} = d_{v_{min}}$). $\mathcal{EX}$ =Expansions$(a_{max}, b'_{max}, g, d_{\rm v})$

Initialization: $a = a_{max} - 1$.
while $a \geq g/2$ **do**
  $\mathcal{EX}_{(a,b)} \leftarrow dot,\ \forall\ b \leq b'_{max}$.
  for $b = 1, \ldots, b'_{max}$ **do**
    $m = 2$.
    while $a + m \leq a_{max}$ **do**
      if $b \leq b'_{max} - 2 + m(d_{\rm v} - 2)$ **then**
        $\mathcal{EX}_{(a,b)} \leftarrow pa_m$.
        if $m \geq g/2$ **then**
          for $c = g/2, \ldots, m$ **do**
            $\mathcal{EX}_{(a,b)} \leftarrow lo_m^c$.
          end for
        end if
      end if
      $m = m + 1$.
    end while
  end for
  $a = a - 1$.
end while
Output: $\mathcal{EX}$.

Algorithm 8. This covers the class of shortest simple cycles of the graph. For irregular graphs, the value is chosen as $b'_{max} = 4$ in Algorithm 8. When the values of a_{max} and b'_{max} are set, the expansions $\mathcal{EX}$ needed for all the relevant classes of LETS structures are determined through Routine 13. This will be used by the *dpl* search algorithm of Chapter 5 or that of Chapter 6, for regular and irregular codes, respectively, to search for an elementary codeword with weight $a \leq a_{max}$. If such a codeword is found, then a is an upper bound for the minimum distance of the code. If the *dpl* search terminated without finding any codeword, or if one is interested in tightening the upper bound, one can increase the value of b'_{max} in a new search, to allow for covering more LETS structures. In the latter case, where an elementary codeword of weight $d^{(u)}_{min}$ has already been found, one should set $a_{max} = d^{(u)}_{min} - 1$, for the new search. Algorithm 8 continues until it provides a satisfactory upper bound

Algorithm 6 Finding an upper bound $d^{(u)}_{min}$ on the minimum distance of an LDPC code.

1: **Initializations:** Set $a_{max} = a_0$, $b_{max} = 0$, $b'_{max} = g/2(d_v - 2)$ for variable-regular codes, or $b'_{max} = 4$ for irregular codes, and $d^{(u)}_{min} = \infty$.
2: $\mathcal{EX}$ =**Expansions**$(a_{max}, b'_{max}, g, d_v)$.
3: Run the *dpl* search based on the expansions in $\mathcal{EX}$.
4: **while** the *dpl* search is running **do**
5: **if** the runtime exceeds T, **then**
6: Stop the search, and go to Step 25.
7: **end if**
8: **if** a LETS in an $(a, 0)$ class is found, where $a < d^{(u)}_{min}$, **then**
9: Stop the search, and set $d^{(u)}_{min} = a$.
10: **end if**
11: **end while**
12: **if** $d^{(u)}_{min} = \infty$, **then**
13: $b'_{max} = b'_{max} + 1$, and go to Step 3.
14: **end if**
15: **if** $d^{(u)}_{min} < \infty$, but looking for a tighter bound, **then**
16: $b'_{max} = b'_{max} + 1$, $a_{max} = d^{(u)}_{min} - 1$, and go to Step 3.
17: **end if**
18: **Output:** $d^{(u)}_{min}$.

on d_{min}, or until the *dpl* search takes too much time to be completed (more than T). In the latter scenario, if the upper bound $d^{(u)}_{min}$ happen to be still at infinity, one can run the algorithm, this time with a larger a_0 and/or T.

Remark 14. *The large difference between the smallest weight of elementary codewords and that of non-elementary codewords (as, e.g., reflected in the results of Table 8.1) suggests that for most LDPC codes, the minimum distance is, in fact, determined by the smallest weight of elementary codewords. Thus, the approach presented here, to focus on the search for elementary codewords, is well-justified both from the computational complexity point of view, and from the viewpoint of being more effective for estimating* d_{min}.

8.4 Numerical results

We have applied our technique to find lower and upper bounds on the minimum distance of a large number of variable-regular and irregular LDPC codes. These include both random and structured codes with a wide range of rates and block lengths. Here, we only present the results for 15 regular and 6 irregular codes. These codes and their parameters can be seen in Tables 8.3 and 8.4, respectively. For all the runtimes reported in this work, a desktop computer with 8 processing cores (2.4-GHz CPU) and 8-GB RAM is used, and the search algorithms are implemented in MATLAB.

In Tables 8.3 and 8.4, for the cases where the exact d_{min} is found, this value is reported in the column corresponding to the lower bound, and we have "-" in the upper bound column. Otherwise, the value L_{ne} is reported as the lower bound, and the upper bound is obtained using the non-exhaustive *dpl* search algorithm. In such cases, the value b'_{max} that has been used to provide the upper bound is reported in the last column of the table. For all cases, the runtime to obtain the lower and upper bounds are also reported. For structured codes, their structural properties are used to simplify the search. These codes are $\mathcal{C}_6, \mathcal{C}_8 - \mathcal{C}_{10}, \mathcal{C}_{12} - \mathcal{C}_{15}$, in Table 8.3, and $\mathcal{C}_{16} - \mathcal{C}_{19}$, in Table 8.4.

We note that the lower bounds (or the exact minimum distances) are all obtained in times that are at most about 30 minutes, and in many cases, only in a few seconds. The upper bounds are obtained in at most about 3 hours, and in many cases, only in a few minutes. The exact minimum distance of C_1 is found in about two minutes. However, using a computer with Pentium-4 3GHz CPU and 512MB RAM, it took the search algorithm of [39], about 44 hours to find that minimum distance.

Table 8.3: Minimum Distance or Bounds on the Minimum Distance of some Variable-Regular LDPC Codes

Code	d_v	Girth	Rate	Length	Lower Bound	Upper Bound	b'_{max}
$\mathcal{C}_1$ [57]	3	6	0.87	495	4 – – – 127 *sec.*	-	-
$\mathcal{C}_2$ [57]	3	6	0.5	504	$d_{min} \geq 8$ – – – 5 *sec.*	20 – – – 8 *min.*	4
$\mathcal{C}_3$ [57]	3	6	0.5	816	$d_{min} \geq 8$ – – – 6 *sec.*	30 – – – 18 *min.*	5
$\mathcal{C}_4$ [57]	3	6	0.77	1057	8 – – – 47 *sec.*	-	-
$\mathcal{C}_5$ [57]	3	6	0.87	16383	$d_{min} \geq 8$ – – – 2053 *sec.*	10 – – – 164 *min.*	5
$\mathcal{C}_6$ [80]	3	8	0.41	155	$d_{min} \geq 12$ – – – 6 *sec.*	20 – – – 1 *min.*	4
$\mathcal{C}_7$ [43]	3	8	0.5	504	$d_{min} \geq 12$ – – – 55 *sec.*	20 – – – 57 *min.*	5
$\mathcal{C}_8$ [85]	3	8	0.88	4000	8 – – – 64 *sec.*	-	-
$\mathcal{C}_9$ [81]	3	8	0.82	5219	12 – – – 455 *sec.*	-	-
$\mathcal{C}_{10}$ [84]	3	10	0.5	546	14 – – – 28 *sec.*	-	-
$\mathcal{C}_{11}$ [69]	3	12	0.5	4896	24 – – – 729 *sec.*	-	-
$\mathcal{C}_{12}$ [85]	4	8	0.69	1274	8 – – – 10 *sec.*	-	-
$\mathcal{C}_{13}$ [85]	4	8	0.77	2890	$d_{min} \geq 16$ – – – 413 *sec.*	20 – – – 8 *min.*	10
$\mathcal{C}_{14}$ [85]	5	8	0.23	210	10 – – – 4 *sec.*	-	-
$\mathcal{C}_{15}$ [85]	5	8	0.75	8000	$d_{min} \geq 20$ – – – 154 *sec.*	-	-

An upper bound of 20 on the minimum distance of C_2 has been reported in the literature [39], [41], [10]. This upper bound matches the one obtained here. Also, the exact minimum distance of C_2 has been reported in [49] to be, in fact, 20. For the purpose of comparing the runtimes, we note that, using a PC with 3.1 GHz CPU and 1GB RAM, it took the algorithm of [49], about 2335 minutes to find the minimum distance of C_2. In comparison, it has taken the non-exhaustive *dpl* search algorithm of this chapter only 13 minutes to find an elementary codeword of weight 20 (minimum weight) for this code.

To the best of our knowledge, the bounds or the exact minimum distances of random codes C_3-C_5 and C_7 have not been reported in the literature so far. We however, believe that the runtimes reported here would be much less than those of any existing search algorithm, including that of [49]. In fact, in our opinion, no existing algorithm would be able to handle C_5, which is a code of rate 0.87 and block length 16383.

Code $\mathcal{C}_6$ is the Tanner (155,64) QC-LDPC code. It takes the proposed algorithms only 6 and 58 seconds to find the lower and upper bounds on the minimum distance of this code, respectively. Our upper bound matches the exact minimum distance of this code, which was found in [81] to be 20, using MAGMA. Code $\mathcal{C}_9$ is a $(5219, 4300)$ QC-LDPC code [81]. To the best of our knowledge, except the analytical upper bound of 24 [81], no other result exists on the minimum distance of this code. It takes the exhaustive *dpl* search algorithm of this work only less than 8 minutes to find the exact minimum distance of this code.

Code $\mathcal{C}_{11}$ is the Ramanujan $(4896, 2448)$ code with $g = 12$. For this code, we find the exact value of d_{min} to be 24, in about 12 minutes. We note that it was reported in [49] that their algorithm terminated without providing tight lower or upper bounds on the minimum distance of this code. Also, in [10], the impulse method was used to find an upper bound of 24 on d_{min} for this code in about 10 hours.

In [84], the authors constructed QC-LDPC codes that are cyclic liftings of fully-connected $3 \times n$ protographs, and have the shortest block length for a given girth. Code C_{10} is the shortest cyclic lifting of the 3×6 fully-connected base graph with girth 10, reported in [84]. We find d_{min} of this code to be 14 in 28 seconds. It took MAGMA a few hours to find this minimum distance [84].

Very recently, QC-LDPC codes with girth 8, whose parity-check matrices have

some symmetries, and are in many cases shorter than previously existing girth-8 QC-LDPC codes, were constructed in [85]. Using MAGMA, the authors obtained and reported the minimum distance of some of the constructed codes in [85]. Only the minimum distances of codes with ($d_{\rm v} = 3$ and $n < 1000$), ($d_{\rm v} = 4$ and $n < 500$), and ($d_{\rm v} = 5$ and $n < 400$) were reported, since for the other codes, MAGMA ran for a day and was still unable to obtain the minimum distance. We tested the codes of [85], and observed that our proposed algorithm can find the exact d_{min}, or obtain lower and upper bounds on d_{min}, for many of them in a matter of seconds or minutes. For example, while in Table I of [85], among 18 codes with $d_{\rm v} = 3$, $g = 8$, $R \leq 0.88$ and $n \leq 4000$, the minimum distance of only the first 11 (those with $R < 0.79$ and $n < 800$) is reported, we have found the minimum distances of all 18 codes, each in less than or about one minute. The last code in that table is C_8 in Table 8.3.

While all the variable-regular codes studied in [49], [41], [10] and [39], have $d_{\rm v} = 3$, the algorithms proposed in this chapter have no such limitation. As an example, we are able to provide lower bounds on, or obtain the exact value of, d_{min} for all the variable-regular LDPC codes provided in Tables II and III of [85], in just a few minutes. These are codes with variable degrees 4 and 5, respectively, and with $R \leq 0.84$ and $n \leq 14750$. In many cases, also, we find upper bounds on d_{min} for these codes. Four examples of the codes in Tables II and III of [85] are listed as the last entries of Table 8.3.

In Table 8.4, we report lower and upper bounds on the minimum distance of six irregular codes. Codes $\mathcal{C}_{16} - \mathcal{C}_{19}$ have been adopted in standards, and Codes $\mathcal{C}_{20}$ and $\mathcal{C}_{21}$ are constructed by the PEG algorithm. The reported upper bounds on d_{min} for $\mathcal{C}_{16} - \mathcal{C}_{19}$ are matched with the upper bounds reported in [70] (no runtime for obtaining these results was reported in [70]). Also, the reported upper bound on d_{min} of $\mathcal{C}_{20}$ is matched with that of [10]. It was reported in [10] that the bound was obtained in a few hours (compared with 21 minutes in this work).

Table 8.4: Minimum Distance or Bounds on the Minimum Distance of some Irregular LDPC Codes

Code	Girth	Rate	Length	Lower Bound	Upper Bound	b'_{max}
$\mathcal{C}_{16}$ [101]	6	0.83	1824	$d_{min} \geq 4$ – – – 7 *sec.*	8 – – – 4 *min.*	7
$\mathcal{C}_{17}$ [101]	6	0.75	2304	$d_{min} \geq 6$ – – – 8 *sec.*	12 – – – 3 *min.*	8
$\mathcal{C}_{18}$ [101]	6	0.67	2304	$d_{min} \geq 4$ – – – 9 *sec.*	15 – – – 3 *min.*	7
$\mathcal{C}_{19}$ [100]	6	0.75	1944	$d_{min} \geq 6$ – – – 12 *sec.*	12 – – – 5 *min.*	8
$\mathcal{C}_{20}$ [43]	8	0.5	1008	$d_{min} \geq 8$ – – – 113 *sec.*	13 – – – 21 *min.*	10
$\mathcal{C}_{21}$ [43]	8	0.5	2048	$d_{min} \geq 8$ – – – 67 *sec.*	15 – – – 20 *min.*	10

Chapter 9

Characterization and Efficient Search of Non-Elementary Trapping Sets of LDPC Codes

9.1 Introduction

In Chapter 7, a lower bound on the size of the smallest elementary and non-elementary trapping sets for a given b in variable-regular LDPC codes is given. Based on the given lower bounds, we demonstrated that the size of the smallest possible non-elementary trapping sets is, in general, larger than that of elementary trapping sets. This provides a theoretical justification as to why non-elementary trapping sets are often not among the most harmful trapping sets. However, the justification provided in Chapter 7, has no claim on the multiplicity of ETSs and NETSs in the classes that both exist. To support the theoretical justification of Chapter 7, using the parent-child relationships between elementary trapping sets and non-elementary trapping sets, we propose an efficient search algorithm to provide an exhaustive/non-exhaustive list of NETSs in the interest range that the ETSs have been already searched for. To the best of our knowledge, the branch-&-bound approach in [97] is the only exhaustive TS search algorithm in the literature. However, similar to the other branch-&-bound algorithms, this approach is only applicable to the short block length codes (the block lengths of all the reported codes are less than 1008).

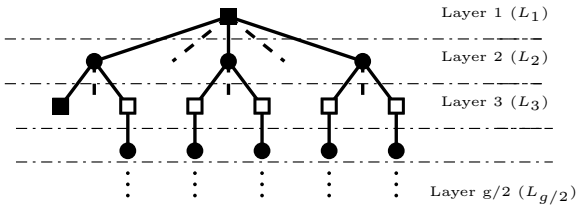


Figure 9.1: A tree-like expansion of a TS rooted at a check node of degree k.

9.2 Exhaustive Search of NETSs in Variable-Regular LDPC Codes

Exhaustive characterization and search algorithms of LETSs in variable-regular and irregular graphs have been proposed in Chapters 5 and 6, respectively. Also in Chapter 6, ETSLs were studied in irregular Tanner graphs. It was shown that these structures can be characterized and searched in any interest range, efficiently and exhaustively. Since variable-regular graphs are a special case of irregular graphs, we thus have similar results for ETSLs in variable-regular LDPC codes and the exhaustive search of ETSLs in Chapter 6 can be used to find them in variable-regular ones as well.

In Chapter 7, based on the presence of a degree-k check node in the induced subgraph of trapping sets, a lower bound on the size of the smallest elementary and non-elementary trapping sets for a given b in variable-regular LDPC codes was derived. From Theorem 5, one can see that for any given b, g and d_{v} values, by increasing k, the size of smallest TS is increased.

Remark 15. *In Table 7.1, it was shown that for any given $b \leq 5$, $d_{\mathrm{v}} = 3, 4, 5, 6$ and $g = 6, 8, 10$, the smallest possible TSs are LETSs.*

Lemma 11. *A lower bound on the size a of an (a, b) non-elementary trapping set in G, whose induced subgraph contains more than one check node of degree k (≥ 3) is greater than or equal to the one provided in Theorem 5.*

Proof. It is easy to show that, following the same tree-like expansion of the induced subgraph of the TS, for any given d_{v}, g and b values, except the first one, the other degree-k check nodes are replaced by degree-1 or degree-2 check nodes at L_3. These new check nodes have more edges than degree-1 and degree-2 check nodes. Therefore,

in the worst case, the same number of variable nodes is needed at L_4. This results in at least the same number of variable nodes in the induced subgraph. ■

Lemma 12. *A lower bound on the size a of an (a, b) non-elementary trapping set in G, whose induced subgraph contains a combination of degree-k and degree-k', where $3 \leq k' < k$ is greater than or equal to the lower bound provided in Theorem 5 with one degree-k check node.*

Proof. Similar to the proof of Lemma 11. ■

The characterization of ETSs (LETSs and ETSLs) for variable-regular graphs is based on normal graph representation of structures. This approach, however, is not applicable to NETSs. In this chapter, to develop the characterization of NETSs, we investigate the parent-child relationships between ETSs and NETSs. As natural candidates for the expansion of ETSs to reach NETSs, we consider *dot*, *path* and *lollipop* expansions. One can see that the application of *path* and *lollipop* expansions to a TS increases the b value of the structure rather rapidly. For NETSs with relatively small b values, we thus limit the expansions to *dot* in the rest of the chapter. Due to the low computational complexity of *dot* expansion, this results in an efficient NETS search algorithm starting from ETSs. Using only the *dot* expansion limits the variety of NETS structures that can be generated starting from ETS structures. In the following, we first discuss the (successive) application(s) of *dot* expansions to ETS structures and then describe the NETS structures that are out of reach.

Suppose that $\mathcal{S}$ is an (a, b) TS structure of variable-regular Tanner graphs with variable degree d_{v}, where $b \geq 1$. The notation dot_m is used for a *dot* expansion with m edges, connecting a new variable node to m check nodes of $\mathcal{S}$. Similar to Chapter 5, we assume that the new variable node in dot_m is connected to at least two check nodes of $\mathcal{S}$, i.e., $2 \leq m \leq d_{\mathrm{v}}$. However, unlike the dot_m used in Chapter 5, the m edges can be connected to both satisfied and unsatisfied check nodes of $\mathcal{S}$. The following result is simple to prove.

Lemma 13. *Suppose that $\mathcal{S}$ is an (a, b) TS structure of variable-regular Tanner graphs with variable degree d_{v}, where $b \geq 1$. Expansion of $\mathcal{S}$ using dot_m, $2 \leq m \leq d_{\mathrm{v}}$, will result in NETS structure(s) in the $(a + 1, b + d_{\mathrm{v}} - 2q)$ class, where $m = p + q$ and $p \geq 0$ and $q \geq 0$ are the number of edges connecting the new variable node to the satisfied and unsatisfied check nodes of $\mathcal{S}$, respectively.*

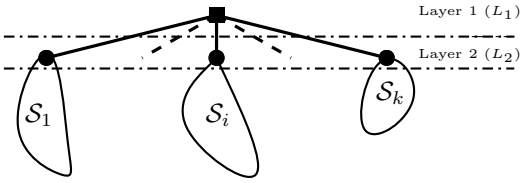


Figure 9.2: A NETS with k disconnected subgraphs.

Remark 16. *One should note that in Lemma 13, if $\mathcal{S}$ is an ETS, then $p > 0$.*

Lemma 14. *Consider the induced subgraph $G(\mathcal{S})$ of a NETS $\mathcal{S}$ in a variable-regular Tanner graph G with variable degree d_{v}. Further, consider the expansions of this subgraph in a layered tree-like fashion starting from one of the check nodes w with degree k, where $k \geq 3$. If any such expansion can be partitioned into k subgraphs, where the only connection of the subgraphs is through w, as shown in Fig. 9.2, then the NETS structure $\mathcal{S}$ cannot be generated by (successive) application(s) of dot_m expansions, $2 \leq m \leq d_{\mathrm{v}}$, to any ETS structure.*

Proof. Consider a NETS structure $\mathcal{S}$ whose subgraph satisfies the condition of the lemma, i.e., there is a tree-like expansion of $G(\mathcal{S})$ rooted at a degree-k check node ($k \geq 3$) with k disconnected subgraphs $\mathcal{S}_1, ..., \mathcal{S}_k$, as shown in Fig. 9.2. In Fig. 9.2, the degree-k check node at the root (first layer L_1) is connected to k variable nodes at layer L_2. Those variable nodes are each connected to $d_{\mathrm{v}} - 1$ other check nodes at L_3 and so on. It is straightforward to see that a structure with the subgraph of Fig. 9.2 cannot be generated through successive applications of $dot_m, m \geq 2$, to an ETS structure $\mathcal{S}'$. The reason is that to create the check node w (with degree $k \geq 3$) in the process of expansion, there are two possibilities: (i) Node w belongs to $\mathcal{S}'$, or (ii) it is added in the expansion process. In Case (i), the degree of w in $\mathcal{S}'$ is either one or two. For the degree of w to be increased to k in the expansion process, through one or more dot_m expansions, one or more variable nodes will have to be added to the subgraph, each with one connection to w and with one or more connection(s) to the other check nodes of the existing (connected) subgraph. This is in contradiction with the structure in Fig 9.2, where otherwise disconnected subgraphs $\mathcal{S}_1, ..., \mathcal{S}_k$ are only connected through w. The proof for Case (ii) is similar. ■

In the following lemma, we investigate the smallest size of NETS structures with induced subgraphs of the form discussed in Lemma 14 and presented in Fig. 9.2, for

Table 9.1: The smallest size a of NETSs in the (a, b) class ($b \leq 4$) with disconnected subgraphs for Tanner graphs with $d_{\mathrm{v}} = 3, 4, 5, 6$, $g = 6, 8, 10$

	$d_{\mathrm{v}} = 3$			$d_{\mathrm{v}} = 4$			$d_{\mathrm{v}} = 5$			$d_{\mathrm{v}} = 6$		
	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$
$b = 1$	15	21	27	not possible			21	39	93	not possible		
$b = 2$	14	20	26	not possible			20	38	92	not possible		
$b = 3$	11	15	19	not possible			19	37	91	not possible		
$b = 4$	10	14	18	15	24	57	18	36	90	21	36	90

different values of d_{v}, g, and $b \leq 4$. Similar results may be derived for other variable degrees, girths and b values.

Lemma 15. *For variable-regular graphs with $d_{\mathrm{v}} = 3, 4, 5, 6$, $g = 6, 8, 10$, and for $b \leq 4$, the size a of the smallest possible NETS in the (a, b) class containing disconnected subgraphs (as shown in Fig. 9.2) is listed in Table 9.1.*

Proof. Suppose that $\mathcal{S}^*$ is the smallest NETS structure with disconnected subgraphs. Based on Corollary 4, the structure $\mathcal{S}^*$ contains a degree-3 check node at L_1. Consider each subgraph $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ as a TS containing the degree-3 check node at L_1 as a degree-1 (unsatisfied) check node (Fig. 9.2). Let $a_{\mathcal{S}_i}$ and $b_{\mathcal{S}_i}$ be the size and the number of unsatisfied check nodes of $\mathcal{S}_i$, respectively. Clearly, $b_{\mathcal{S}_i} > 0$. We also have

$$b_{\mathcal{S}_1} + b_{\mathcal{S}_2} + b_{\mathcal{S}_3} = b + 2, \tag{9.1}$$

For $\mathcal{S}^*$ to have the smallest size, we look for $\mathcal{S}_i$s with smallest $a_{\mathcal{S}_i}$ whose $b_{\mathcal{S}_i}$ values satisfy $b_{\mathcal{S}_i} > 0$, and the constraint (9.1). It is easy to see that the most favorable candidate for TSs $\mathcal{S}_i$s is an ETSL with no cycle. These structures are denoted by $ETSL_2$, and exist only in the $(a, b = a(d_{\mathrm{v}} - 2) + 2)$ class. If an $ETSL_2$ structure is not possible (due to the specific choice of $b_{\mathcal{S}_i}$), then based on Remark 15, a LETS structure is the next favorable choice. To find the size of $\mathcal{S}^*$, therefore, one needs to consider all the possible combinations of positive integers $b_{\mathcal{S}_1}$, $b_{\mathcal{S}_2}$ and $b_{\mathcal{S}_3}$ that satisfy (9.1), and for each combination finds the smallest values of $a_{\mathcal{S}_i}$s using the aforementioned guidelines. In the following, we prove the result for the case of $d_{\mathrm{v}} = 3$, $g = 8$ and $b = 2, 3$. The proof for the other cases listed in Table 9.1 is similar.

For $b = 2$, using (9.1), we have $b_{\mathcal{S}_1} + b_{\mathcal{S}_2} + b_{\mathcal{S}_3} = 4$. The only positive integers satisfying this equality are $\{1, 1, 2\}$. Since, for $d_{\text{v}} = 3$, there does not exist any $ETSL_2$ with $b < 3$, then we look for LETS structures of minimum size with these b values. From Table 7.1 in Chapter 7, the size of the smallest LETSs with $b_{\mathcal{S}_i} = 1$ and $b_{\mathcal{S}_i} = 2$ is 7 and 6, respectively. We thus conclude that the size of $\mathcal{S}^*$ is $7 + 7 + 6 = 20$.

For $b = 3$, we have $b_{\mathcal{S}_1} + b_{\mathcal{S}_2} + b_{\mathcal{S}_3} = 5$. The only solutions to this equation are $\{1, 2, 2\}$ and $\{1, 1, 3\}$. Again for the first set, no $ETSL_2$ structure exists, and based on LETS structures of minimum size, we obtain $7 + 6 + 6 = 19$ as the size of the corresponding NETS structure. For the second set of $b_{\mathcal{S}_i}$ values, we select an $ETSL_2$ structure for the $b_{\mathcal{S}_i}$ value 3. This corresponds to $a_{\mathcal{S}_i} = 1$. For the other two TSs, the minimum size LETS structures have size 7, and thus the size of the corresponding NETS structure in this case is $1 + 7 + 7 = 15$. Since 15 is the smaller value between 19 and 15, it is in fact the size of $\mathcal{S}^*$.

To obtain the entries in Table 9.1 for even values of d_{v}, one should note that based on Lemma 1, it is not possible to have a TS with even d_{v} and odd b. Therefore, for even values of d_{v}, the minimum value of $b_{\mathcal{S}_i}$ is 2, and in (9.1), the smallest value of b for NETS structures under consideration is strictly larger than 3. ■

In the following, we investigate the parent-child relationships between ETSs and NETSs based on dot_m expansions. Since the NETS structures discussed in Lemmas 14 and 15 are excluded, in the rest of this chapter, we use the expression "interest range of a and b" or "(a, b) class of interest" to mean the b values that satisfy $b \leq 4$, and for a given b value in this range, the value of a being strictly less than the entry provided in Table 9.1.

Proposition 14. *Any NETS structure $\mathcal{S}$ of variable-regular graphs with variable degree d_{v} in an (a, b) class of interest, containing only one degree-3 check node (the rest of the check nodes have degree 2 or 1) can be characterized by the application of a dot_m expansion ($2 \leq m \leq d_{\text{v}}$) to an ETS substructure, $\mathcal{S}'$, in the $(a-1, b+2m-2-d_{\text{v}})$ class, where m is number of edges connecting the variable node in $\mathcal{S}/\mathcal{S}'$ to one degree-2 and $m-1$ degree-1 check nodes of $\mathcal{S}'$.*

Proof. The structure $\mathcal{S}$ contains only one check node w of degree 3. We consider the tree-like expansion of $\mathcal{S}$ from w as the root at L_1. Based on the knowledge that this expansion of $\mathcal{S}$ does not consist of disconnected subgraphs as shown in Fig. 9.2, there must exist two variable nodes, say v_1 and v_2 at L_2 that are connected through a

path that does not pass through w. Now, consider removing one of these two variable nodes, say v_1, and all its incident edges from $\mathcal{S}$. The remaining graph $\mathcal{S}'$ is still connected and has no check node with degree larger than 2, i.e., $\mathcal{S}'$ is an ETS. It is easy to see that $\mathcal{S}$ can be obtained by expanding $\mathcal{S}'$ by v_1 through a dot_m $(2 \leq m \leq d_{\rm v})$ expansion. The class of $\mathcal{S}'$ can be obtained by using Lemma 13, assuming $p = 1$. ■

The following corollary describes the exhaustive search of NETSs with only one degree-3 check node.

Corollary 7. *In variable-regular Tanner graphs with variable degree $d_{\rm v}$, all the (a, b) NETSs containing only one degree-3 check node in the interest range of $a \leq a_{\max}$ and $b \leq b_{\max}$ ($a_{\max}$ less than the value in Table 9.1 and $b_{max} \leq 4$) can be found by applying dot_m expansions to all the ETSs in the range of $a \leq a_{\max} - 1$ and $b \leq b'_{\max} = b_{\max} + d_{\rm v} - 2$.*

Proposition 15. *Any NETS structure $\mathcal{S}$ in an interest class of (a, b) for variable-regular graphs with variable degree $d_{\rm v}$, that contains two degree-3 check nodes (the rest are degree-2 or -1) can be characterized by a dot_m expansion $(2 \leq m \leq d_{\rm v})$ applied to one of the two following substructures $\mathcal{S}'$: (i) an ETS in the $(a-1, b+2m-4-d_{\rm v})$ class, where from m edges connecting the variable node in $\mathcal{S}/\mathcal{S}'$ to $\mathcal{S}'$, two and $m-2$ are connected to degree-2 and degree-1 check nodes of $\mathcal{S}'$, respectively; or (ii) a NETS containing one degree-3 check node in the $(a-1, b+2m-2-d_{\rm v})$ class, where from m edges connecting the variable node in $\mathcal{S}/\mathcal{S}'$ to $\mathcal{S}'$, one and $m-1$ are connected to degree-2 and degree-1 check nodes of $\mathcal{S}'$, respectively.*

Proof. Consider the expansion of the NETS structure starting from one of the degree-3 check nodes w at L_1. In the expansion, the other degree-3 check node is located either at L_3 or at L_{2i+1}, where $i > 1$. With an argument similar to the one presented in the proof of Proposition 14, there exist two variable nodes v_1 and v_2 at L_2 such that there is a path between them that does not pass through w. Therefore, by removing one of these two variable nodes, say, v_1, and all its incident edges from $\mathcal{S}$, the resulted subgraph $\mathcal{S}'$ remains connected. Now if the second degree-3 check node was at L_3 and connected to v_1, then there remains no check node with degree larger than 2 after the removal of v_1, i.e., the subgraph $\mathcal{S}'$ is an ETS. On the other hand, if the other degree-3 check node was at L_3 but not connected to v_1 or it was at L_{2i+1} with $i > 1$, then the resulted subgraph $\mathcal{S}'$ is a NETS containing one degree-3 check node. In either case, structure $\mathcal{S}$ can be obtained by applying a dot_m expansion $(2 \leq m \leq d_{\rm v})$

to $\mathcal{S}'$. The class of $\mathcal{S}'$ can be determined in each case by using Lemma 13, assuming $p = 2$ and $p = 1$, respectively. ■

The following result is a generalization of Proposition 15.

Proposition 16. *Any NETS structure $\mathcal{S}$ in an interest class of (a, b) for variable-regular graphs with variable degree d_v, that contains $f \geq 2$ degree-3 check nodes (the rest are degree-2 or -1) can be characterized by a dot_m expansion ($2 \leq m \leq d_v$) applied to one of the $\eta = \min\{m, f\}$ following substructures $\mathcal{S}'$: For any value of p in the range $1 \leq p \leq \eta$, substructure $\mathcal{S}'$ is in the $(a-1, b+2(m-p)-d_v)$ class, where from m edges connecting the variable node in $\mathcal{S}/\mathcal{S}'$ to $\mathcal{S}'$, p and $m-p$ are connected to degree-2 and degree-1 check nodes of $\mathcal{S}'$, respectively.*

Based on the above results, it is easy to see that a NETS structure with f degree-3 check nodes can be generated through successive applications of dot_m expansions to ETS structures. For this to correspond to an exhaustive search of such NETSs, the following corollary, that generalizes Corollary 7, provides the range of ETSs that need to be included.

Corollary 8. *In variable-regular Tanner graphs with variable degree d_v, all the NETSs containing f degree-3 check nodes (the rest are degree-2 or -1) in the interest range of $a \leq a_{\max}$ and $b \leq b_{\max}$ ($a_{\max}$ less than the value in Table 9.1 and $b_{max} \leq 4$) can be found by f successive applications of dot_m expansions to all the ETSs in the range of $a \leq a_{\max} - \lceil f/d_v \rceil$ and $b \leq b'_{\max} = b_{\max} + f(d_v - 2)$.*

The following results can all be proved similar to the cases involving NETSs with only degree-3 check nodes. The proofs are thus omitted to avoid redundancy.

Proposition 17. *Any NETS structure $\mathcal{S}$ in an interest class of (a, b) for variable-regular graphs with variable degree d_v, that contains only one degree-4 check node (the rest are degree-2 or -1) can be characterized by a dot_m expansion ($2 \leq m \leq d_v$) applied to a NETS substructure $\mathcal{S}'$ containing only one degree-3 check node in the $(a-1, b+2m-d_v)$ class. From m edges connecting the variable node in $\mathcal{S}/\mathcal{S}'$ to $\mathcal{S}'$, one and $m-1$ are connected to degree-3 and degree-1 check nodes of $\mathcal{S}'$, respectively.*

Corollary 9. *In variable-regular Tanner graphs with variable degree d_v, all the NETSs containing only one degree-4 check node (the rest are degree-2 or -1) in the interest range of $a \leq a_{\max}$ and $b \leq b_{\max}$ ($a_{\max}$ less than the value in Table 9.1 and*

$b_{max} \leq 4$) can be found by two successive applications of dot_m expansions to all the ETSs in the range of $a \leq a_{\max} - 2$ and $b \leq b'_{\max} = b_{\max} + 2d_{\mathrm{v}} - 2$.

Proposition 18. *Any NETS structure $\mathcal{S}$ in the interest class of (a, b) for variable-regular graphs with variable degree d_{v}, that contains one degree-4 and one degree-3 check nodes (the rest are degree-2 or -1) can be characterized by a dot_m expansion ($2 \leq m \leq d_{\mathrm{v}}$) applied to one of the following substructures $\mathcal{S}'$: (i) a NETS substructure, containing only one degree-3 check node, in the $(a-1, b+2m-2-d_{\mathrm{v}})$ class, where out of $m \geq 2$ edges connecting the variable node in $\mathcal{S}/\mathcal{S}'$ to $\mathcal{S}'$, one is connected to a degree-3 check node, one to a degree-2 check node and $m-2$ to degree-1 check nodes of $\mathcal{S}'$. (ii) a NETS substructure, containing two degree-3 check nodes, in the $(a-1, b+2m-d_{\mathrm{v}})$ class, where out of $m \geq 2$ edges connecting the variable node in $\mathcal{S}/\mathcal{S}'$ to $\mathcal{S}'$, one and $m-1$ are connected to degree-3 and degree-1 check nodes of $\mathcal{S}'$, respectively.*

Corollary 10. *In variable-regular Tanner graphs with variable degree d_{v}, all the NETSs containing one degree-4 and one degree-3 check nodes (the rest are degree-2 or -1) in the interest range of $a \leq a_{\max}$ and $b \leq b_{\max}$ ($a_{\max}$ less than the value in Table 9.1 and $b_{max} \leq 4$) can be found by three successive applications of dot_m expansions to all the ETSs in the range of $a \leq a_{\max} - 2$ and $b \leq b'_{\max} = b_{\max} + 3d_{\mathrm{v}} - 4$.*

Remark 17. *Note that in all cases discussed in Corollaries 7-10, successive applications of dot_m expansions to the ETSs can result in finding structures in addition to the ones that are of interest in these corollaries.*

Corollaries 7-10 demonstrate that by increasing the multiplicity of check nodes with degrees larger than 2 and the degrees of such check nodes, the range of b values for ETSs that are needed to provide an exhaustive search of such NETSs is increased. To have an efficient NETS search algorithm based on successive dot_m expansions of ETSs, we limit the multiplicity and the degrees of such check nodes to the following cases in the rest of this chapter: NETSs containing at most four degree-3 check nodes, or containing only one degree-4 and at most one degree-3 check nodes. We use notations $N_3, N_{3,3}, N_{3,3,3}$, and $N_{3,3,3,3}$, to denote NETS structures with only one up to four check nodes of degree 3. Notations N_4 and $N_{4,3}$ are used for NETS structures that contain only one degree-4 check node and those with only one degree-4 and one degree-3 check nodes, respectively.

Corollary 11. *In variable-regular Tanner graphs with variable degree d_{v}, all the N_3 ,$N_{3,3}$,$N_{3,3,3}$,$N_{3,3,3,3}$, N_4, and $N_{4,3}$ in the interest range of $a \leq a_{\max}$ and $b \leq b_{\max}$ ($a_{\max}$ less than the value in Table 9.1 and $b_{max} \leq 4$) can be found by up to four successive applications of dot_m expansions to all the ETSs in the range of $a \leq a_{\max} - 1$ and $b \leq b'_{\max} = b_{\max} + \max\{(3d_{\mathrm{v}} - 4), (4d_{\mathrm{v}} - 8)\}$.*

By restricting the NETS structures to those discussed above, we limit the maximum size a_{max} of NETSs that can be exhaustively covered. The following theorem provides the value of a_{max} for Tanner graphs with different d_{v} and g values.

Theorem 8. *For a variable-regular Tanner graph with variable-degree d_{v} and girth g, consider the union of sets N_3 ,$N_{3,3}$,$N_{3,3,3}$,$N_{3,3,3,3}$, N_4, and $N_{4,3}$, obtained by successive applications of dot_m expansions ($m \geq 2$) to ETSs within the range indicated in Corollary 11. For $d_{\mathrm{v}} = 3, 4, 5, 6$, and $g = 6, 8, 10$, Table 9.2 provides the value of $a_{\max}$ such that such a union gives an exhaustive list of NETSs of the Tanner graph within the range of $a \leq a_{max}$ and $b \leq b_{max} = 4$.*

Proof. Based on the sets of NETSs that are covered, it is easy to see that the exhaustive search is limited by the size of the smallest structure in sets $N_{3,3,3,3,3}$, $N_{4,3,3}$, $N_{4,4}$ and N_5. The structures in $N_{3,3,3,3,3}$, however, have $b \geq 5$, and thus not in the range of interest of the theorem. We thus find the size $a^*_{4,3,3}$, $a^*_{4,4}$ and a^*_5 (or a lower bound on the size) of the smallest structure in sets $N_{4,3,3}$, $N_{4,4}$ and N_5, respectively, and list $a_{max} = a^* - 1$ in Table 9.2, where $a^* = \min\{a^*_{4,3,3}, a^*_{4,4}, a^*_5\}$.

For structures in N_5, we use Theorem 5 with different values of $b \leq 4$, and choose the smallest lower bound as a^*_5. For structures in $N_{4,3,3}$ and $N_{4,4}$, we use the tree-like expansion of the NETS structure, starting from a degree-4 check node at the root in L_1. The tree thus has four variable nodes in L_2. The idea is to grow this tree into a NETS structure of smallest size with no cycle of length smaller than g and with the given b value, where out of b unsatisfied check nodes in the case of $N_{4,3,3}$, two of them have degree 3. To minimize the size, one needs to select the check nodes to have the minimum degree within the above constraints. For structures in $N_{4,3,3}$, this means selecting all the satisfied check nodes (other than the root) to have degree 2 and all the $b - 2$ unsatisfied check nodes to have degree 1. For structures in $N_{4,4}$, it means that all the satisfied check nodes, except for the root and one other check node with degree 4, the rest must have degree 2. The b unsatisfied check nodes in this case all have degree 1. To satisfy the girth constraint, all the variable and check

nodes in the first $g/2$ layers of the tree must be distinct (i.e., no cycle should appear in the subgraph). Moreover, in the tree, there are four subgraphs, each starting from one variable node at L_2. To avoid having cycles shorter than g in these subgraphs, any new variable (check) node at $L_{g/2+1}$, for $g/2$ odd (even), can only be connected to the check (variable) nodes of each such subgraph at most once. Therefore, for odd values of $g/2$, at $L_{g/2+1}$, we need, at least, as many variable nodes as the number of edges emanating from the check nodes at $L_{g/2}$ of each subgraph to $L_{g/2+1}$. Also, for even values of $g/2$, if the number of variable nodes in a subgraph at $L_{g/2}$ times $d_{\mathrm{v}}-1$ is larger than the number of the rest of variable nodes at $L_{g/2}$ (in the other $k-1$ subgraphs), more variable nodes are needed to be added at $L_{g/2+2}$ to complete the connections required for check nodes at $L_{g/2+1}$.

Considering the above constraints, for both cases of structures in $N_{4,3,3}$ and $N_{4,4}$, and for each value of b, we find the structure with the smallest number of variable nodes. The values $a^*_{4,3,3}$ and $a^*_{4,4}$ are then obtained by taking the minimum among the smallest sizes corresponding to five different values of $b = 0, \ldots, 4$. In the following, we discuss in more details, the proof for one entry of Table 9.2. Proofs for other entries are similar.

Consider Tanner graphs with $d_{\mathrm{v}} = 3$, $g = 8$ and NETSs with $b_{\max} = 4$. Based on Theorem 5, we have $a^*_5 = 12$.

For $N_{4,3,3}$, to minimize the size of a NETS, the two degree-3 check nodes must be located at L_3 and be connected to two different variable nodes at L_2. This minimizes the number of variable nodes needed in the higher layers of the tree while satisfying the girth constraint. All the remaining $b-2$ unsatisfied check nodes with degree 1 must also be located at L_3. The remaining check nodes at L_3 are thus $8-b$ degree-2 check nodes. This means there must be $4+(8-b)$ variable nodes at L_4, and $4+4+8-b = 16-b$ variable nodes in the whole structure up to L_4. It appears that by proper addition of check nodes in L_5, no more variable node is needed in L_6. The smallest size of structures in $N_{3,3,4}$ for different values of $b \leq 4$ is thus obtained by $b = 4$, and we have $a^*_{4,3,3} = 12$. As two examples, the smallest NETS structures for $b = 3$ and $b = 4$ are given in Fig. 9.3.

For $N_{4,4}$, to minimize the size of NETS, the second degree-4 check node must be located at L_3. All the b degree-1 check nodes must also be at L_3. Out of $4(d_{\mathrm{v}}-1) = 8$ check nodes in L_3, one is degree-4, b are degree-1 and $7-b$ are degree-2. This means there are $3+(7-b)$ variable nodes at L_4. If $b \geq 1$, to satisfy the girth constraint with

Table 9.2: The maximum size $a_{\max}$ of (a, b) NETSs that can be searched exhaustively within the range $b \leq b_{max} = 4$ by successive applications of dot_m expansions to (a', b') ETSs with size up to $a_{max} - 1$ and $b' \leq b'_{\max}$. (The lower bound on the size of smallest possible NETS with $b \leq 4$ is given in brackets.)

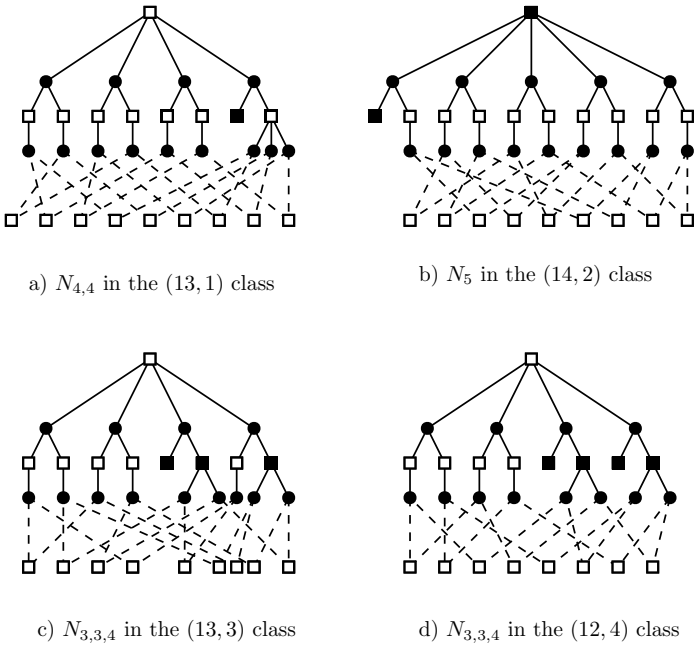
	$d_v = 3$			$d_v = 4$			$d_v = 5$			$d_v = 6$		
	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$
$a_{\max}$	6(4)	11(6)	16(8)	6(5)	15(9)	24(15)	8(6)	19(12)	43(24)	8(7)	21(15)	46(35)
$b'_{\max}$	9	9	9	12	12	12	16	16	16	20	20	20

minimum number of variable nodes, one degree-1 check node in L_3 is connected to the same variable node in L_2 that has also a connection to the degree-4 check node in L_3. If $b > 1$, the rest of degree-1 check node(s) are each connected to another (different) variable node in L_3. Now, for $b > 1$, consider a variable node v_1 in L_2 that is connected to one degree-4 and one degree-1 check node in L_3 and call the subtree rooted at v_1 as subtree 1. This subtree has 3 variable nodes at L_4 that must be connected to $3(d_v - 1) = 6$ distinct check nodes at L_5. To complete the connections of these check nodes, at least six variable nodes should exist at L_4 of the rest of the subtrees (excluding subtree 1), otherwise, more variable nodes are needed at L_6. By considering all the cases of $b \leq 4$, we conclude that $a^*_{4,4} = 13$. As two examples, the smallest NETS structures for $b = 1$ and $b = 4$ are shown in Fig. 9.3.

Based on the above, for $d_v = 3$, $g = 8$, we have $a_{\max} = \min\{12, 12, 13\} - 1 = 11$. ■

Using Corollary 11, one can find the $b'_{\max}$ value which indicates the range of b values for ETSs that are required for the exhaustive search of the desired NETSs. The $b'_{\max}$ values for graphs with different d_v values are also provided in Table 9.2. In Table 9.2, we have also included the lower bound on the size of the smallest possible NETS with $b \leq 4$ in brackets. As an example, the entries corresponding to $d_v = 3$, and $g = 8$ in Table 9.2 show that, for such variable-regular graphs, we can exhaustively search all the NETSs with $a = 6, 7, 8, 9, 10, 11$ and $b \leq 4$.

The pseudo-code of the proposed search algorithm is given in Algorithm 7. In the proposed *dot-based* NETS search algorithm, the input is the exhaustive list of ETSs in the range of $a \leq a_{max} - 1$ and b'_{max}. In the search process, *dot* expansion is applied to any instance of TSs (ETSs and NETSs) in the interest range of $a \leq a_{max} - 1$ and $b \leq b'_{max}$. The sets $\mathcal{I}^a_{TS}$ and $\mathcal{I}^a_{ETS}$ are the sets of all the instances of TSs and ETSs



a) $N_{4,4}$ in the $(13,1)$ class

b) N_5 in the $(14,2)$ class

c) $N_{3,3,4}$ in the $(13,3)$ class

d) $N_{3,3,4}$ in the $(12,4)$ class

Figure 9.3: Four examples of smallest possible NETS structures in $N_{4,3,3}$ and $N_{4,4}$ for variable-regular graphs with $d_{\mathrm{v}} = 3$, $g = 8$, in the range $b \leq 4$.

in the (a, b) classes with $b \leq b'_{\max}$, respectively. The set $\mathcal{I}^{a}_{NETS}$ is the set of all the instances of NETSs in the (a, b) classes with $b \leq b_{\max}$.

Remark 18. *We note that if in Algorithm 7, we increase the value of a_{max} beyond that of Table 9.2 (but less than the one in Table 9.1), by exhaustive search of ETSs in the range of $a \leq a_{max}$ and $b \leq b'_{max}$, we can still find all the $N_3, N_{3,3}, N_{3,3,3}, N_{3,3,3,3}, N_4, N_{4,3}$ structures in the new range of $a \leq a_{max}$ and $b \leq 4$, but there is no guarantee to find the other NETS structures in the new range exhaustively.*

9.3 Non-Exhaustive Search of NETSs in Variable-Regular LDPC Codes

The exhaustive search of NETSs proposed in Section 9.2 has two limitations. First, the value of $b'_{\max}$ obtained in Section 9.2, is rather large which implies a high complexity for the exhaustive search of ETSs. Moreover, for the given values of d_{v}, g and $b_{\max}$, the value of $a_{\max}$ is relatively small. For these two reasons, we propose a non-exhaustive search of NETSs in a wider range of a and b values based on setting

Algorithm 7 (NETS Search) Finds list of the instances of (a, b) NETS structures of a variable-regular Tanner graph $G = (V, E)$ with girth g and variable degree d_{v}, for $a \leq a_{\max}$ and $b \leq b_{max}$ ($a_{\max}$ is obtained from Table 9.2 for $b_{max} = 4$). The input is all the instances of (a, b) ETS structures, in the range $a \leq a_{\max} - 1$ and $b \leq b'_{\max}$, $\mathcal{I}_{ETS}$ ($b'_{\max}$ is obtained from Table 9.2). The output is the set $\mathcal{I}_{NETS}$, which contains the instances of NETSs in the interest range. $\mathcal{I}_{NETS}$= **NetsSrch** $(\mathcal{I}_{ETS}, a_{max}, b_{max})$

Inputs: $G, g, d_{\mathrm{v}}, \mathcal{I}_{ETS}$ (a_{max}, $b_{max} = 4$).
Initializations: $\mathcal{I}^a_{NETS} \leftarrow \emptyset; \mathcal{I}^a_{TS} = \mathcal{I}^a_{ETS}$, $\forall\ a \leq a_{\max}$; $b'_{\max}$ is obtained from Table 9.2.
for $a = g/2, \ldots, a_{\max} - 1$ **do**
    for any structure $\mathcal{S} \in \mathcal{I}^a_{TS}$ **do**
        Consider $\mathcal{V}$ to be the set of variable nodes in $V \setminus \mathcal{S}$ which have at least two connections to the check nodes in $\Gamma(\mathcal{S})$.
        for each variable node $v \in \mathcal{V}$ **do**
            $\mathcal{S}' = \{\mathcal{S} \cup v\} \setminus \mathcal{I}^{a+1}_{TS}$.
            $b = |\Gamma_o(\mathcal{S}')|$.
            if $b \leq b'_{\max}$ **then**
                $\mathcal{I}^{a+1}_{TS} = \mathcal{I}^{a+1}_{TS} \cup \mathcal{S}'$.
                if $b \leq b_{\max}$ **then**
                $\mathcal{I}^{a+1}_{NETS} = \mathcal{I}^{a+1}_{NETS} \cup \mathcal{S}'$.
                end if
            end if
        end for
    end for
end for
Output: $\mathcal{I}_{NETS} = \{\mathcal{I}^a_{NETS},\ \forall\ a \leq a_{\max}\}$.

$b'_{\max} = b_{\max} + t$, where $t \geq 1$, instead of the value indicated in Table 9.2. Our experimental results show that by increasing $b'_{\max}$ beyond $b_{\max} + 2$, the number of new NETSs that can be found in the interest range is negligible.

The NETS search algorithm proposed in Algorithm 7 can also be used for the non-exhaustive search of NETSs. As the input, in this case, one should find and provide all the ETSs in the range $a \leq a_{max}$ and $b \leq b'_{max}$. However, since the b'_{max} is less than the value given in Table 9.2, the list of NETSs, $\mathcal{I}_{NETS}$, would be non-exhaustive. One should also note that since the algorithm imposes no restriction on the degree of check nodes of searched NETSs, by increasing $a_{\max}$, some other NETSs with combination of different check node degrees can be found as well.

9.4 Non-Exhaustive Search of NETSs in Irregular LDPC Codes

Due to the variety of variable degrees in variable-irregular LDPC codes, we are not able to provide results similar to those in Section 9.2 in relation to exhaustive search of NETSs in irregular codes. Algorithm 7 can, however, be still used for the non-exhaustive search of NETSs in irregular graphs. To obtain an exhaustive list of ETSs as the input to Algorithm 7 in this case, one can use the search algorithms of Chapter 6.

9.5 Numerical Results

Table 9.3 shows the list of TSs of Tanner (155,64) code [82] with $d_{\text{v}} = 3$ and $g = 8$ in the wide range of $a \leq 13$ and $b \leq 4$. The distinguished numbers of LETS, ETSL and NETS structures are shown as well. One can compare the number of LETS, ETSL and NETS in this code to see that in the classes believed to be potentially harmful (with relatively small a and b), the only TSs are LETSs. Based on Table 9.2, the results of NETSs for the classes with $a \leq 11$ and $b \leq 4$ are exhaustive. For finding NETSs beyond that range, in Algorithm 7, the exhaustive list of ETSs within the range of $a \leq 12$ and $b \leq 5$ has been used. In [97], authors used the branch-& -bound approach to propose an exhaustive search algorithm to find all the TSs. However, similar to the other branch-& -bound algorithms, this approach is only applicable to

Table 9.3: Multiplicities of (a, b) TSs consisting of LETSs, ETSLs and NETSs of Tanner (155,64) within the range of $a \leq 13$ and $b \leq 4$

	Tanner (155,64)				
(a, b) class	Total LETS	Total ETSL	Total NETS	Total TS	Total TS [97]
(4,4)	465	0	0	465	465
(5,3)	155	0	0	155	155
(6,4)	930	1860	0	2790	2790
(7,3)	930	0	0	930	930
(8,2)	465	0	0	465	465
(8,4)	5115	9300	0	14415	14415
(9,3)	1860	3720	0	5580	5580
(10,2)	1395	0	0	1395	1395
(10,4)	29295	48360	5580	83235	83235
(11,3)	6200	9300	1860	17360	17360
(12,2)	930	0	0	930	930
(12,4)	180885	134850	47895	363630	**36280**
(13,3)	34875	5580	2790	43245	43245

the short block length codes. The number of TSs in different classes in this range found by our algorithm for Tanner (155,64) code is matched with the one reported in [97] except for the $(12, 4)$ class. While in *dpl* search algorithm, 363630 TSs have been found in the $(12, 4)$ class, only 36280 TSs have been reported in [97].[1] The run-time of the proposed algorithm in [97] to find the exhaustive list of TSs is not reported. However, while it took the algorithm of [97], 59 minutes[2] to find the TSs of a PEG code in the interest range of $a \leq 5$ and $b \leq 5$, our *dpl* search algorithm finds all of them in less than 20 seconds.

This search of NETSs along with the theoretical results in Chapter 7 support the assertion that in the harmful classes of TSs, there is no NETSs (otherwise, they could potentially be harmful).

[1] We believe that the result reported in [97] should be a typographical error.

[2] This is the only run-time reported in [97]. The run-time is for a standard desktop PC with a 2.67-GHz processor.

Chapter 10

Tight Lower and Upper Bounds on the Stopping Distance of LDPC Codes

10.1 Introduction

In this chapter, we use the graphical structure of stopping sets within the Tanner graph of an LDPC code to devise our search algorithm, and to derive bounds on the stopping distance, $s_{\min}$ of LDPC codes. The subgraph induced by a stopping set in the Tanner graph of a code contains only check nodes with degrees two or more. We consider two categories of stopping set depending on the check node degrees in their subgraph. If all the check nodes have degree two, we call the stopping set *elementary*. Otherwise, the stopping set is referred to as *non-elementary*. Considering that an elementary stopping set is in fact an elementary codeword, we can use the approach proposed in Chapter 8 to find them. Moreover, the non-elementary stopping sets are a subset of non-elementary trapping sets discussed in Chapter 9. Using the exhaustive/non-exhaustive search of LETSs and NETSs, one can provide some lower and upper bounds on the stopping distance of LDPC codes. The general approach is similar to the one for minimum distance in Chapter 8. However, unlike the search of codewords, stopping sets which are non-elementary are considered as well. Moreover, to have an efficient search algorithm, we limit the search to find NESSs containing small-degree check nodes, thus, the search is exhaustive in a certain range.

10.2 Lower Bound on the Stopping Distance of Variable-Regular LDPC Codes

By definition, an ESS is a trapping set for which the degree of all the check nodes are 2, thus ESSs and elementary codeword, discussed in Chapter 8 are identical. Therefore, an ESS of an LDPC code corresponds to a leafless elementary trapping set (LETS) in the code's Tanner graph with $b = 0$. The following lower bound on stopping distance is simple to prove.

Proposition 19. *The result of Theorem 5 with $k = 2$ and $b = 0$ provides a lower bound, L_{SS_1}, on $s_{\min}$ for variable-regular LDPC codes.*

Remark 19. *We note that the result of Proposition 19 (Not Theorem 5) is essentially the same as the lower bound obtained in [64] on the stopping distance of variable-regular LDPC codes.*

To potentially improve the lower bound of Proposition 19, L_{SS_1}, we use the fact that ESSs, as a special case of LETSs, have a graphical structure that lends itself well to the efficient exhaustive *dpl* search algorithm of Chapter 5. Using the *dpl* search algorithm with $b_{\max} = 0$, we can efficiently and exhaustively find all the ESSs of a variable-regular LDPC code with a maximum given size $a_{\max}$. In the following, we establish a lower bound, L_{SS_2} ($L_{SS_2} > L_{SS_1}$) , on the size of smallest NESSs. We then perform an exhaustive *dpl*-based search of ESSs of maximum size $a_{\max} = L_{SS_2} - 1$. If this search does not find any ESS, then we establish $L_{SS_2} \leq s_{\min}$. Otherwise, the smallest size of found ESSs is the exact value of $s_{\min}$.

Proposition 20. *The result of Theorem 5 with $k = 3$ and $b = 1$ provides a lower bound, L_{SS_2}, on the size of non-elementary stopping sets.*

To further improve the lower bound of Proposition 20 on $s_{\min}$, if possible, we need to perform an exhaustive search of NESSs. This can be performed, by using the NETS search algorithm of Chapter 9 with some modifications as described below.

We first note that, compared to Chapter 9, here, we are not interested in NETSs with unsatisfied check nodes of degree-1. This implies that the range of exhaustive search for NESSs, as a subset of NETSs, can be potentially increased.

We use notations SS_3, $SS_{3,3}$, $SS_{3,3,3}$, $SS_{3,3,3,3}$, to denote NESSs with only one up to four check nodes of degree 3, respectively. Notations SS_4 and $SS_{4,3}$ are used

Table 10.1: The maximum size $a_{\max}$ of SSs that can be searched exhaustively by successive applications of dot_m expansions to (a', b') LETSs with size up to $a_{max} - 1$ and $b' \leq b'_{\max}$.

	$d_v = 3$			$d_v = 4$			$d_v = 5$			$d_v = 6$		
	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$
$a_{\max}$	7	13	19	8	17	29	8	19	43	10	22	56
$b'_{\max}$	9	9	9	12	12	12	16	16	16	20	20	20

for NESSs that contain only one degree-4 check node and only one degree-4 and one degree-3 check nodes, respectively. Similar to Chapter 9, we limit the search of NESSs to the following configurations: SS_3,$SS_{3,3}$, $SS_{3,3,3}$, $SS_{3,3,3,3}$, SS_4, and $SS_{4,3}$. The following result is in parallel with Corollary 11.

Corollary 12. *In variable-regular Tanner graphs with variable degree d_v, all the SS_3,$SS_{3,3}$, $SS_{3,3,3}$, $SS_{3,3,3,3}$, SS_4, and $SS_{4,3}$ in the interest range of $a \leq a_{\max}$ ($a_{\max}$ less than the value in Table 9.1) can be found by up to four successive applications of dot_m expansions to all the LETSs in the range of $a \leq a_{\max} - 1$ and $b \leq b'_{\max} = 4 + \max\{(3d_v - 4), (4d_v - 8)\}$.*

The following result (parallel to Theorem 8) provides the range in which the NESS search is exhaustive.

Theorem 9. *For a variable-regular Tanner graph with variable-degree d_v and girth g, consider the union of sets SS_3,$SS_{3,3}$, $SS_{3,3,3}$, $SS_{3,3,3,3}$, SS_4, and $SS_{4,3}$, obtained by successive applications of dot_m expansions ($m \geq 2$) to LETSs within the range indicated in Corollary 12. For $d_v = 3, 4, 5, 6$, and $g = 6, 8, 10$, Table 10.1 provides the value of $a_{\max}$ such that such a union gives an exhaustive list of NESSs of the Tanner graph within the range of $a \leq a_{max}$.*

Proof. The proof is similar to the proof of Theorem 8 with the following differences: (i) Despite the case of Theorem 8, where NETSs with only $b \leq 4$ were studied, here, for NESSs, there is no such limitation, and we are interested in NESSs with any value of b (including those with $b > 4$), (ii) Since there exists no degree-1 check node in a SS, the degree of all the unsatisfied check nodes of NESSs should be odd values greater than or equal to 3, (iii) Due to (i), in addition to minimum-size structures in $SS_{4,3,3}$, $SS_{4,4}$, SS_5, one needs to also consider the minimum-size structures in $SS_{3,3,3,3,3}$ as

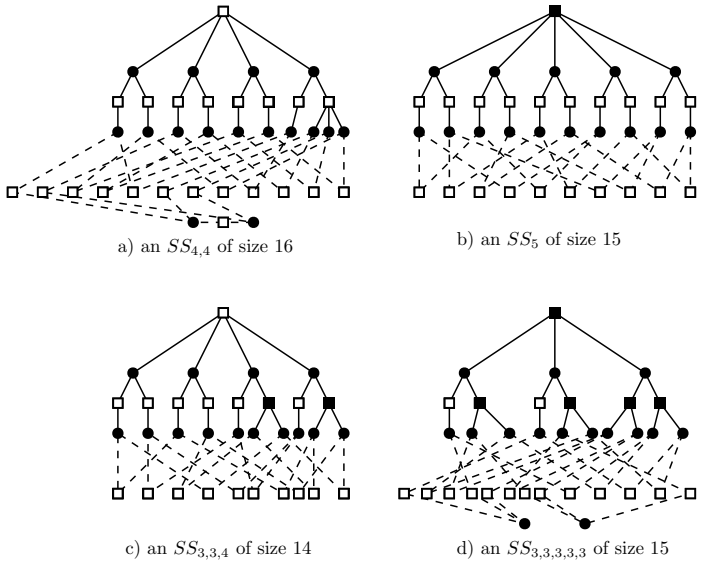


Figure 10.1: Four examples of smallest NESS structures in variable-regular graphs with $d_{\mathrm{v}} = 3, g = 8$.

potentially limiting the range of exhaustive search, (*iv*) In the tree-like expansion of the subgraph, to minimize the size of the NESS and due to the non-existence of degree-1 check nodes, one needs to assume that except the few check nodes with degree larger than 2, all the rest of check nodes have degree-2. ■

Fig. 10.1 shows four examples of smallest NESS structures of $SS_{4,3,3}$, $SS_{4,4}$, SS_5 and $SS_{3,3,3,3,3}$ for graphs with $d_{\mathrm{v}} = 3$ and $g = 8$.

Remark 20. *Note that the values of a_{max} in Table 10.1 are generally larger than those in Table 9.2. For example, while the range of exhaustive search of NESSs for variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 8$ is* 13*, this value for the exhaustive search of NETSs is* 11*.*

If the exhaustive search of SSs (ESSs and NESSs) up to size a_{max} listed in Table 10.1 fails to find any SS, then $L_{SS_3} = a_{max} + 1$ is a lower bound on $s_{\min}$. Otherwise, the smallest size of found SSs gives the exact value of $s_{\min}$.

In Table 10.2, we have listed L_{SS_3}, as well as the values of L_{SS_1} and L_{SS_2}, obtained from Propositions 19 and 20, for Tanner graphs with different values of d_{v} and g.

Example 28. *In variable-regular graphs with $d_{\mathrm{v}} = 3$ and $g = 8$, while the size of smallest possible ESSs (from Proposition 19) is* 6*, by the exhaustive search of ESSs,*

Table 10.2: Values L_{SS_3},L_{SS_2} and L_{SS_1} (as potential lower bounds on s_{min}) for codes with $d_v = 3, 4, 5, 6$, and $g = 6, 8, 10$

	$d_v = 3$			$d_v = 4$			$d_v = 5$			$d_v = 6$		
	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$	$g = 6$	$g = 8$	$g = 10$
L_{SS_3}	8	14	20	9	18	30	9	20	44	11	23	57
L_{SS_2}	5	9	13	7	12	21	7	15	31	8	18	43
L_{SS_1}	4	6	10	5	8	17	6	10	26	7	12	37

one can potentially improve the lower bound on s_{min} to 9*. Moreover, by considering NESSs, the bound can be potentially improved further to* 14*.*

The pseudo code for obtaining a lower bound $s^{(l)}_{\min}$ on $s_{\min}$ is presented in Algorithm 8. The algorithm starts by exhaustively searching for ESSs of size at most $L_{SS_3} - 1$ in Lines 5-9. During the search of ESSs, if any ESS is found, the size of that ESS is assigned as the temporary value for $s^{(l)}_{min}$. (Since the search of LETSs is hierarchical, if any SS is found in Line 6, it is the smallest one in the range of interest.) If this temporary $s^{(l)}_{min}$ is larger than L_{SS_2}, then the NETS search algorithm of Chapter 9 (Algorithm 7) is used to find NESSs with size less than this temporary $s^{(l)}_{min}$. If such a NESS is found, the size of that NESS is assigned as the new and final value for $s^{(l)}_{min}$. (Since the search of NETSs is hierarchical, if any NESS is found in Line 13, it is the smallest one in the range of interest.)

Remark 21. *In Chapter 9, the input of Algorithm 7 for the search of NETSs in the range of $a \leq a_{max}$ and $b \leq 4$ was the list of all ETSs in the range of $a \leq a_{max} - 1$ and $b \leq b'_{max}$. However, NESSs are leafless, i.e., each variable node is connected to at least two check nodes. Therefore, for finding the NESSs in Line 13 of Algorithm 8, the input is just the list of LETSs in the range of $a \leq a_{max} - 1$ and $b \leq b'_{max}$, that has already been found in Line 4.*

Remark 22. *We note that by removing the conditions that stop the algorithm when an ESS or a NESS is found, one can find the list of all stopping sets with size less than or equal to $a_{max} = L_{SS_3} - 1$ exhaustively.*

Algorithm 8 Finding a lower bound $s_{\min}^{(l)}$ on the stopping distance of a variable-regular Tanner graph G with variable degree d_{v} and girth g.

Inputs: G, g, d_{v}.
Initializations: Set $a_{\max} = L_{SS_3}$, and select $b'_{\max}$ from Table 10.1.
$s_{\min}^{(l)} = a_{\max}$.
Run the exhaustive LETS search algorithm in the range of $a \leq a_{max} - 1$ and $b \leq b'_{max}$.
while the *dpl* search is running **do**
    if a LETS in an $(a, 0)$ class is found **then**
        Stop the search, and set $s_{\min}^{(l)} = a$, $a_{\max} = a$.
    end if
end while
if $s_{\min}^{(l)} > L_{SS_2}$ **then**
    $\mathcal{I}_{NETS}$= **NetsSrch**($\mathcal{I}_{LETS}$, a_{max}, $b_{max} = 4$)
    while the NETS search is running **do**
        if a NESS of size a is found, where $a < a_{\max}$, **then**
            Stop the search, and set $s_{\min}^{(l)} = a$.
        end if
    end while
end if
Output: $s_{\min}^{(l)}$.

10.3 Upper Bound on the Stopping Distance of LDPC Codes

In this part, we consider the non-exhaustive search of SSs for both variable-regular and irregular LDPC codes. If we fail to find the exact stopping distance of an LDPC code (variable-regular) based on the approach described in Section 10.2, then, we have established that $s_{\min} \geq L_{SS_3}$. For such cases, in this section, we also find an upper bound on $s_{\min}$. To obtain this upper bound, we find a stopping set with weight larger than L_{SS_3}. The approach is similar to the one provided in Section 8.3 for minimum distance of LDPC codes. However, in addition to searching for LETSs (Section 8.3), the NETS search is used here to find a low weight stopping set.

A pseudo code for obtaining an upper bound $s_{\min}^{(u)}$ on stopping distance is presented in Algorithm 9. To start the algorithm, one can select $a_{\max}$ to be initially a rather large value a_0, say three or four times the value of L_{SS_3}. The procedure for searching the ESSs and NESSs are similar to the one in Section 10.2. There are only two differnces. First, in the exhaustive search of LETSs (Line 4 of Algorithm 8), the set of expansions $\mathcal{EX}$, and b'_{max} are obtained by the characterization Algorithm 1. Second, in the exhaustive search of NETSs (Line 11 of Algorithm 8) in the range of $a \leq a_{max}$ and $b \leq 4$, the value of b'_{max} for the exhaustive search of LETSs is from Table 10.1. However, here, for both non-exhaustive search of LETSs and NETSs, $b'_{\max}$ is chosen to be a rather small value. This value for variable-regular codes is set at $b'_{\max} = g/2(d_v - 2)$, in Algorithm 9. This covers the class of shortest simple cycles of the graph. Also, for the search of NETSs (NESSs) in Algorithm 7, $b_{max} = b'_{max}$. For irregular graphs, the value is chosen as $b'_{\max} = 4$ in Algorithm 9. Also, when the values of $a_{\max}$ and $b'_{\max}$ are set, the expansions $\mathcal{EX}$ needed for all the relevant classes of LETS structures are determined through Routine 13. If an ESS of size a is found, then a is a temporary upper bound for the stopping distance of the code. Then the NETS search is used to find any possible NESS with size less than the size of the smallest ESS. If such a NESS is found, then its size is an upper bound on the stopping distance of the code. If the search terminated without finding any stopping set, or if one is interested in tightening the upper bound, one can increase the value of $b'_{\max}$ in a new search, to allow for covering more structures. In the latter case, where a stopping set of weight $s_{\min}^{(u)}$ has already been found, one should set $a_{\max} = s_{\min}^{(u)} - 1$, for the new search.

Algorithm 9 Finding an upper bound $s_{\min}^{(u)}$ on the stopping distance of Tanner graph G with variable degree d_{v} and girth g.

1: **Inputs:** G, g, d_{v}.
2: **Initializations:** Set $a_{\max} = a_0$, $b'_{\max} = g/2(d_{\mathrm{v}} - 2)$ for variable-regular codes, or $b'_{\max} = 4$ for irregular codes, and $s_{\min}^{(u)} = \infty$.(For irregular codes, $d_{\mathrm{v}} = d_{\mathrm{v}_{\min}}$.)
3: $\mathcal{EX}$ =**Expansions**$(a_{\max}, b'_{\max}, g, d_{\mathrm{v}})$ (Routine 13).
4: Run Algorithm 1 for the LETS search based on the expansions in $\mathcal{EX}$.
5: **while** the LETS search is running **do**
6: **if** the run-time exceeds T, **then**
7: Stop the LETS search, and go to Step 25.
8: **end if**
9: **if** a LETS in an $(a, 0)$ class is found, where $a < s_{\min}^{(u)}$, **then**
10: Stop the LETS search, set $s_{\min}^{(u)} = a$.
11: **end if**
12: **end while**
13: $\mathcal{I}_{NETS}$= **NetsSrch**$(\mathcal{I}_{LETS}, a_{\max}, b_{\max} = 4)$
14: **while** the NETS search is running **do**
15: **if** a NESS of size a is found, where $a < s_{\min}^{(u)}$, **then**
16: Stop the NETS search, set $s_{\min}^{(l)} = a$.
17: **end if**
18: **end while**
19: **if** $s_{\min}^{(u)} = \infty$, **then**
20: $b'_{\max} = b'_{\max} + 1$, and go to Step 3.
21: **end if**
22: **if** $s_{\min}^{(u)} < \infty$, but looking for a tighter bound, **then**
23: $b'_{\max} = b'_{\max} + 1$, $a_{\max} = s_{\min}^{(u)} - 1$, and go to Step 3.
24: **end if**
25: **Output:** $s_{\min}^{(u)}$.

10.4 Numerical results

We have applied our technique to find lower and upper bounds on the stopping distance of a large number of variable-regular and irregular LDPC codes. These include both random and structured codes with a wide range of rates and block lengths. Here, we present the results for 20 variable-regular and 8 irregular codes. These codes and their parameters can be seen in Tables 10.3 and 10.4. In Tables 10.3 and 10.4, for the cases where the exact $s_{\min}$ is found, this value is reported in the column corresponding to the lower bound, and we have "-" in the upper bound column. Otherwise, the value L_{SS_3} is reported as the lower bound, and the upper bound is obtained using the non-exhaustive search algorithms. In such cases, the value $b'_{\max}$ that has been used to provide the upper bound is reported in the last column of the table. For all cases, the run-time to obtain the lower and upper bounds are also reported. For structured codes, their structural properties are used to simplify the search. These codes are $\mathcal{C}_{10}, \mathcal{C}_{13} - \mathcal{C}_{20}$ in Table 10.3, and $\mathcal{C}_{21} - \mathcal{C}_{26}$, in Table 10.4. Also, for all cases, the letter e or n is reported in brackets to indicate whether the smallest SS found in the search algorithm is elementary or non-elementary, respectively.

We note that the lower bounds (or the exact stopping distances) are all obtained in times that are at most about 16 minutes, and in many cases, only in a few seconds. The upper bounds are obtained in at most about 35 minutes, and in many cases, less than 4 minutes. Using a computer with Core 2 Duo E6700 2.67GHz CPU and 2 GB of RAM, it took the search algorithm of [40], about 600 and 3085 hours to provide an upper bound on the stopping distance of $\mathcal{C}_1$ and $\mathcal{C}_3$, respectively. In comparison, it has taken our non-exhaustive *dpl* search algorithm only 5 and 34 minutes to find the same upper bounds for $\mathcal{C}_1$ and $\mathcal{C}_3$, respectively. Also the exact stopping distance of $\mathcal{C}_1$ has been reported in [70], which is matched with the bound reported here (the run-time has not been reported in [70]).

To the best of our knowledge, the upper bound for random codes C_2 and $\mathcal{C}_4$ have not been reported in the literature so far. It takes our algorithm only 43 seconds to find the exact stopping distance of $\mathcal{C}_4$, a random high rate code with block length 1057. We believe that the run times reported here would be much less than those of any existing search algorithm. In fact, in our opinion, no existing algorithm would be able to handle $\mathcal{C}_9$, which is a code of rate 0.87 and block length 16383. It takes about only 2 and 3 minutes to provide the lower and upper bounds of 7 and 9 for this

Table 10.3: Stopping Distance or Bounds on the Stopping Distance of some Variable-Regular LDPC Codes

Code	d_v	Girth	Rate	Length	Lower Bound	Upper Bound	$b'_{\max}$
$\mathcal{C}_1$ [57]	3	6	0.5	504	$s_{\min} \geq 7$ – – – 5 *sec.*	$16(n)$ – – – 5 *min.*	4
$\mathcal{C}_2$ [57]	3	6	0.5	816	$s_{\min} \geq 7$ – – – 6 *sec.*	$24(n)$ – – – 6 *min.*	4
$\mathcal{C}_3$ [57]	3	6	0.5	1008	$s_{\min} \geq 7$ – – – 6 *sec.*	$26(n)$ – – – 34 *min.*	5
$\mathcal{C}_4$ [57]	3	6	0.77	1057	$7(n)$ – – – 43 *sec.*	-	-
$\mathcal{C}_5$ [58]	3	6	0.75	2000	$6(e)$ – – – 15 *sec.*	-	-
$\mathcal{C}_6$ [58]	3	6	0.77	3000	$s_{\min} \geq 7$ – – – 48 *sec.*	$8(e)$ – – – 1 *min.*	4
$\mathcal{C}_7$ [58]	3	6	0.8	5000	$6(e)$ – – – 28 *sec.*	-	-
$\mathcal{C}_8$ [58]	3	6	0.81	8000	$s_{\min} \geq 7$ – – – 51 *sec.*	$14(e)$ – – – 23 *min.*	4
$\mathcal{C}_9$ [57]	3	6	0.87	16383	$s_{\min} \geq 7$ – – – 209 *sec.*	$9(n)$ – – – 3 *min.*	3
$\mathcal{C}_{10}$ [80]	3	8	0.41	155	$s_{\min} \geq 13$ – – – 6 *sec.*	$18(n)$ – – – 45 *sec.*	4
$\mathcal{C}_{11}$ [43]	3	8	0.5	504	$s_{\min} \geq 13$ – – – 363 *sec.*	$19(n)$ – – – 10 *min.*	5
$\mathcal{C}_{12}$ [43]	3	8	0.5	1008	$s_{\min} \geq 13$ – – – 245 *sec.*	$37(n)$ – – – 35 *min.*	5
$\mathcal{C}_{13}$ [85]	3	8	0.88	4000	$8(e)$ – – – 73 *sec.*	-	-
$\mathcal{C}_{14}$ [81]	3	8	0.82	5219	$12(e)$ – – – 913 *sec.*	-	-
$\mathcal{C}_{15}$ [84]	3	10	0.5	546	$14(e)$ – – – 42 *sec.*	-	-
$\mathcal{C}_{16}$ [69]	3	12	0.5	4896	$24(e)$ – – – 729 *sec.*	-	-
$\mathcal{C}_{17}$ [85]	4	8	0.69	1274	$8(e)$ – – – 10 *sec.*	-	-
$\mathcal{C}_{18}$ [85]	4	8	0.77	2890	$s_{\min} \geq 17$ – – – 468 *sec.*	$20(e)$ – – – 9 *min.*	10
$\mathcal{C}_{19}$ [85]	5	8	0.23	210	$10(e)$ – – – 4 *sec.*	-	-
$\mathcal{C}_{20}$ [85]	5	8	0.75	8000	$s_{\min} \geq 19$ – – – 154 *sec.*	-	-

code, respectively. Codes $\mathcal{C}_5$-$\mathcal{C}_8$ are 4 high rate random codes with variable degree 3 and girth 6 constructed by the program of [58].[1] These random high rate codes with large block lengths are challenging codes for all the existing approaches in the literature. One can see that the exact stopping distance or the lower and upper bounds of these codes have been reported in most cases in a few seconds. To the best of our knowledge, except a few structured medium block length codes with rate 0.5, no result has been reported in the literature for the relatively large block length and high rate codes.

Also, an upper bound of 18 on the stopping distance of $\mathcal{C}_{10}$ (Tanner (155,64)) has been found in just 45 seconds which matches the exact value reported in [70]. Among the stopping distance of 7 variable-regular LDPC codes reported in [70], the run time of only two structured small block length codes has been reported. The stopping distance of $\mathcal{C}_{10}$ has been found in about 1 minute on a standard desktop computer [70].

Codes $\mathcal{C}_{11}$ and $\mathcal{C}_{12}$ are two variable-regular codes constructed by PEG algorithm [43] (available in [57]). It takes only 10 and 35 minutes to find upper bounds of 19 and 37 on the stopping distance of $\mathcal{C}_{11}$ and $\mathcal{C}_{12}$, respectively. For the purpose of comparing the run-times, we note that, it took the algorithm of [40], about 25 hours to find the same upper bound for $\mathcal{C}_{11}$. This bound also matches the exact stopping distance reported in [70]. Also, to the best of our knowledge, the upper bound 37 of $\mathcal{C}_{12}$ has not been reported in the literature so far. Moreover, the exact stopping distance of $\mathcal{C}_{14}$, a high rate structured codes with block length 5219 has been found in just 913 seconds.

In [84], the authors constructed QC-LDPC codes that are cyclic liftings of fully-connected $3 \times n$ protographs, and have the shortest block length for a given girth. Code C_{15} is the shortest cyclic lifting of the 3×6 fully-connected base graph with girth 10, reported in [84]. We find $s_{\min}$ of this code to be 14 in 28 seconds. Code $\mathcal{C}_{16}$ is the Ramanujan $(4896, 2448)$ code with $g = 12$. For this code, we find the exact value of $s_{\min}$ to be 24, in about 12 minutes. For the purpose of comparing the run-times, we note that, it took the algorithm of [40], about 162 hours to find the same upper bound for $\mathcal{C}_{16}$.

We tested the codes of [85], and observed that our proposed algorithm can find the exact $s_{\min}$, or obtain lower and upper bounds on $s_{\min}$, for many of them in a

[1] Using *code6.c* with seed= 380.

matter of seconds or minutes. For example, we have found the stopping distances of all 18 codes with $d_{\rm v} = 3$, $g = 8$, $R \leq 0.88$ and $n \leq 4000$ (Table I of [85]), each in less than or about one minute. The last code in that table is $\mathcal{C}_{13}$ in Table 10.3.

While most of the variable-regular codes studied in the literature, see, e.g., [70], have $d_{\rm v} = 3$, the algorithms proposed here can find the exact stopping distance, or provide lower and upper bounds on stopping distance of variable-regular codes with $d_{\rm v} > 3$. As an example, we are able to provide lower bounds on, or obtain the exact value of, $s_{\min}$ for all the variable-regular LDPC codes provided in Tables II and III of [85], in just a few minutes. These are codes with variable degrees 4 and 5, respectively, and with $R \leq 0.84$ and $n \leq 14750$. In many cases, also, we find upper bounds on $s_{\min}$ for these codes. Four examples of the codes in Tables II and III of [85] are listed as the last entries of Table 10.3.

Based on the size of stopping distance, block length, rate and degree distribution of the reported codes in the literature [70], [68], [42], [40], we believe finding the exact (or bounds on the) stopping distance of codes such as $\mathcal{C}_3$, $\mathcal{C}_8$, $\mathcal{C}_9$, $\mathcal{C}_{12}$, $\mathcal{C}_{14}$, $\mathcal{C}_{18}$ and $\mathcal{C}_{20}$ are out of the reach of their algorithms or the run times will be significantly larger than ours.

In Section 9.4, we discussed why no lower bound is provided for irregular LDPC codes and in Section 10.3, the search algorithm is applicable to both variable-regular and irregular LDPC codes. We have used this search algorithm to provide an upper bound on the stopping distance of eight irregular codes listed in Table 10.4. Codes $\mathcal{C}_{21} - \mathcal{C}_{26}$ have been adopted in standards, and Codes $\mathcal{C}_{27}$ and $\mathcal{C}_{28}$ are random codes constructed by the PEG algorithm. Authors in [71] reported the exact stopping distance of all the IEEE 802.16e LDPC codes [101]. Our upper bound search algorithm can find the same stopping distance for each of them, most of the time in just a few seconds (no run-time for obtaining these results was reported in [71]).

Table 10.4: Upper Bounds on the Stopping Distance of some Irregular LDPC Codes

Code	Girth	Rate	Length	Upper Bound	$b'_{\max}$
$\mathcal{C}_{21}$ [106]	6	0.5	128	$11(n)$ – – – 118 *sec.*	8
$\mathcal{C}_{22}$ [100]	6	0.83	648	$7(n)$ – – – 24 *sec.*	7
$\mathcal{C}_{23}$ [101]	6	0.83	1824	$8(e)$ – – – 203 *sec.*	7
$\mathcal{C}_{24}$ [101]	6	0.75	2304	$12(e)$ – – – 165 *sec.*	8
$\mathcal{C}_{25}$ [101]	6	0.67	2304	$15(e)$ – – – 184 *sec.*	7
$\mathcal{C}_{26}$ [100]	6	0.75	1944	$12(e)$ – – – 289 *sec.*	8
$\mathcal{C}_{27}$ [43]	8	0.5	1008	$13(e)$ – – – 21 *min.*	10
$\mathcal{C}_{28}$ [43]	8	0.5	2048	$15(e)$ – – – 20 *min.*	10

Chapter 11

Conclusion and Future Work

11.1 Conclusion

In this thesis, we studied all the problematic graphical structures which play important roles in the error floor performance and error correction capability of LDPC codes.

First, we complemented LSS characterization of LETSs. Then we proposed a novel hierarchical graph-based expansion approach to characterize LETSs of variable-regular LDPC codes. The proposed characterization is based on three basic expansion techniques, dubbed, *dot*, *path* and *lollipop*, used in the space of normal graphs. Each LETS structure $\mathcal{S}$ is characterized as a sequence of embedded LETS structures that starts from a simple cycle, and grows in each step by using one of the three expansions, until it reaches $\mathcal{S}$. It was demonstrated that the new characterization, called *dpl*, is minimal, in that, none of the expansions in the sequence can be divided into smaller expansions and still maintain the property that all the embedded substructures are LETSs. Moreover, it was proved that any minimal characterization based on embedded LETS structures must only use one of the three expansions, *dot*, *path* and *lollipop*, at each step. The minimality of the characterization, allowed us to devise search algorithms that are provably efficient in finding all the instances of (a, b) LETS structures with $a \leq a_{max}$ and $b \leq b_{max}$, for any choice of a_{max} and b_{max}, in a guaranteed fashion. The efficiency of the search is a consequence of the fact that to enumerate the instances of larger LETS structures, the proposed algorithm searches for instances of smaller LETS structures that are of interest themselves, i.e., their a and b values are in the range of interest. This is unlike the so-called LSS-based search algorithm of [46], where a large number of instances of LETS structures with large

b values need to be enumerated, not because they are of direct interest, but because they happen to be parents of other LETS structures of interest.

We proposed a hierarchical graph-based approach in the space of normal graphs to characterize ETSs of irregular LDPC codes. Two characterizations were proposed both based on three simple expansion techniques, called *dot*, *path* and *lollipop*. The proposed dpl characterizations, both, describe an ETS as an embedded sequence of ETS structures that starts from a simple cycle or a single variable node, and is expanded step by step through a combination of the three expansions to reach the ETS under consideration. The proposed characterizations/search algorithms can be considered as the generalization of our work on LETSs of variable-regular LDPC codes to the cases where the code has variable nodes with a variety of degrees, and to ETSs that are not leafless.

Moreover, we derived a lower bound on the size of non-elementary trapping sets (NETS) of variable-regular LDPC codes. Using this bound, we demonstrated that the size of the smallest possible NETSs is, in general, larger than that of ETSs with the same number of unsatisfied check nodes. This provides a theoretical justification for why NETSs are often less harmful than ETSs. To further support this claim, using the parent-child relationships between ETSs and NETSs, we proposed an efficient search algorithm to provide an exhaustive/non-exhaustive list of NETSs in the interest range that the ETSs have been already searched for.

We derived tight lower and upper bounds on the minimum distance d_{min} of LDPC codes. The bounds, which were established using a combination of analytical results and search techniques, are applicable to both regular and irregular LDPC codes with a wide range of rates and block lengths. To derive the bounds, we partitioned the non-zero codewords into two categories of elementary and non-elementary, with the former category being those with only degree-2 check nodes in their subgraph. We noted that elementary codewords are LETSs, and as a result lend themselves to efficient *dpl* search algorithms. We then established a lower bound L_{ne} on the weight of non-elementary codewords, and proposed an exhaustive *dpl* search of elementary codewords with weight a in the range $a \leq L_{ne}$. If the search happened to find an elementary codeword, then the smallest weight of such a codeword was d_{min}. Otherwise, if the search failed, then a lower bound of $L_{ne} \leq d_{min}$ was established on d_{min}. For the upper bound, the *dpl* search of LETSs was modified to increase the range of search for elementary codewords with larger weights at the expense of losing

the exhaustiveness of the search.

Also, we derived tight lower and upper bounds on the stopping distance s_{min} of LDPC codes. Similar to the minimum distance, the bounds were established using a combination of analytical results and search techniques. We partitioned the stopping sets into two categories of elementary stopping sets (ESSs) and non-elementary stopping sets (NESSs). We noted that ESSs and NESSs are subsets of LETSs and NETSs, respectively and as a result lend themselves to the efficient *dpl*-based LETS and NETS search algorithms. If the exhaustive search of LETSs and NETSs happened to find a stopping set, then the smallest weight of such a stopping set was s_{min}. Otherwise, if the search failed, then a lower bound was established on s_{min}. For the upper bound, the *dpl* search of LETSs and NETSs was modified to increase the range of search for stopping sets with larger weights at the expense of losing the exhaustiveness of the search.

To the best of our knowledge, the proposed *dpl-based* algorithms presented in this work are the most efficient exhaustive/non-exhaustive search algorithms available for finding the error-prone structures of LDPC codes. It is also the most general, in that, it is applicable to codes with any degree distribution, girth, rate and block length.

The results of this thesis have been published or submitted for publication in [28], [29], [30], [31], [32], [33], [34], [35], [36], [37].

11.2 Future Work

Insights gained from this work can enable further advancement of analysis and design of different types of LDPC codes. As an example, a list of elementary trapping sets can be used as the input of an importance sampling algorithm to estimate the performance of LDPC codes in the error floor region accurately. Another example is using this list for improving the performance of iterative decoders using some post processing. Due to the low computational complexity of proposed *dpl* search algorithms, they can be embedded in an efficient LDPC code search procedure, to find codes that are free of small error-prone structures. Moreover, the approaches provided here can be used in the analysis of the harmful trapping sets of *non-binary LDPC* and *spatially coupled LDPC* codes, as well.

List of References

[1] S. Abu-Surra, D. DeClercq, D. Divsalar, and W. E. Ryan, "Trapping set enumerators for specific LDPC codes," in *Proc. Inf. Theory Appl. Workshop (ITA), 2010*, SanDiego, CA, USA, Jan./Feb. 2010, pp. 1–5.

[2] R. Asvadi, A. H. Banihashemi, and M. Ahmadian-Attari, "Lowering the error floor of LDPC codes using cyclic liftings," *IEEE Trans. Inf. Theory*, vol. 57, no. 4, pp. 2213–2224, Apr. 2011.

[3] R. Asvadi, A. H. Banihashemi and M. Ahmadian-Attari, "Design of finite-length irregular protograph codes with low error floors over the binary-input AWGN channel using cyclic liftings," *IEEE Trans. Comm.*, vol. 60, no. 4, pp. 902 - 907, Apr. 2012.

[4] A. N. Barreto, B. Faria, E. Almeida, I. Rodriguez, M. Lauridsen, R, Amorim, and R. Vieira, "5G Wireless Communications for 2020," *Journal of Comm. and Inf. systems*, vol. 31, no. 1, pp. 146–163, 2016.

[5] B. Butler and P. Siegel, "Error floor approximation for LDPC codes in the AWGN channel," *IEEE Trans. Inf. Theory*, vol. 60, no. 12, pp. 7416–7441, Dec. 2014.

[6] B. Butler, and P. Siegel, " Bounds on the minimum distance of punctured quasi-cyclic LDPC codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 7, pp. 4584–4597, Jul. 2013.

[7] E. Cavus and B. Daneshrad, "A performance improvement and error floor avoidance technique for belief propagation decoding of LDPC codes," in *Proc. 16th IEEE Int. Symp. Personal, Indoor Mobile Radio Commun.*, Los Angeles, CA, USA, Sep. 2005, pp. 2386–2390.

[8] C. Cole, S. Wilson, E. Hall, and T. Giallorenzi, "A general method for finding low error rates of LDPC codes," [Online]. Available: http://arxiv.org/abs/cs/0605051.

[9] A. Dehghan and A. H. Banihashemi, " On the Tanner graph cycle distribution of random LDPC, random protograph-based LDPC, and random quasi-cyclic LDPC code ensembles," submitted to *IEEE Trans. Inf. Theory*, Jan. 2017, available online at: *https://arxiv.org/abs/1701.02379*

[10] D. Declercq, and M. Fossorier, " Improved impulse method to evaluate the low weight profile of sparse binary linear codes," in *Proc. IEEE Int. Symp. on Inf. Theory (ISIT)*, Toronto, Canada, Jul. 2008, pp. 1963–1967.

[11] D. Declercq, B. Vasic, S. K. Planjery, and E. Li, "Finite alphabet iterative decoders, part ii: Towards guaranteed error correction of LDPC codes via iterative decoder diversity," *IEEE Trans. Commun.*, vol. 61, no. 10, pp. 4046–4057, Oct. 2013.

[12] C. Di, D. Proietti, I. E. Telatar, T. J. Richardson, and R. L. Urbanke,"Finite length analysis of low-density parity-check codes," in *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[13] Q. Diao, Y. Y. Tai, S. Lin, and K. Abdel-Ghaffar, "Trapping set structure of finite geometry LDPC codes," in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, Davis, CA, USA, Jul. 2012, pp. 3088–3092.

[14] M. Diouf, S. OUYA, and B. Vasic, "A PEG-like LDPC code design avoiding short trapping sets," in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, Hong King, China, June 2015, pp. 1079–1083.

[15] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "Analysis of absorbing sets for array-based LDPC codes," in *Proc. Int. Conf. Commun. (ICC)*, Glasgow, Scotland, Jun. 2007, pp. 6261–6268.

[16] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, and M. Wainwright, "Analysis of absorbing sets for array-based LDPC codes," in *Proc. Int. Conf. Commun. (ICC)*, Glasgow, Scotland, Jun. 2007, pp. 6261–6268.

[17] L. Dolecek, Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets of array-based LDPC codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 181–201, Jan. 2010.

[18] I. Dumer, D. Micciancio, and M. Sudan, "Hardness of approximating the minimum distance of linear code," *IEEE Trans. Inf. Theory*, vol. 49, no. 1, pp. 2237, Jan. 2003.

[19] M. Esmaeili and M. J. Amoshahy, "On the stopping distance of array code parity-check matrices," *IEEE Trans. Inf. Theory*, vol. 55, no. 8, pp. 3488–3493, Aug. 2009.

[20] J. L. Fan, "Array-codes as low-density parity-check codes," in *Proc. 2nd Int. Symp. Turbo Codes and Related Topics*, Brest, France, Sep. 2000, pp. 543–546.

[21] Y. Fang, P. Chen, L. Wang, and F. C. M. Lau, "Design of Protograph LDPC Codes for Partial Response Channels," *IEEE Trans. Commun.*, vol. 60, no. 10, pp. 2809-2819, Oct. 2012.

[22] R. G. Gallager, " Low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[23] H. Falsafain and S.R. Mousavi, "Exhaustive enumeration of elementary trapping sets of an arbitrary Tanner graph," *IEEE Comm. Lett.*, vol. 20, pp. 1713–1716, Sep. 2016.

[24] H. Falsafain and S.R. Mousavi, "Stopping set elimination by parity-check matrix extension via integer linear programming," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, pp. 1533–1540, May. 2015.

[25] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788-1793, Aug. 2004.

[26] A. Hareedy, C. Lanka, and L. Dolecek, "A General Non-Binary LDPC Code Optimization Framework Suitable for Dense Flash Memory and Magnetic Storage," *IEEE Journal on Selected areas in Comm.*, vol. 34, no. 9, pp. 2402–2415, Sep. 2016.

[27] Y. Han and W. E. Ryan, "LDPC decoder strategies for achieving low error floors," in *Proc. Inform. Theory Appl. Workshop*, San Diego, CA, USA, Jan. 2008, pp. 277–286.

[28] Y. Hashemi and A. Banihashemi, "On characterization and efficient exhaustive search of elementary trapping sets of variable-regular LDPC codes," *IEEE Comm. Lett.*, vol. 19, pp. 323–326, Mar. 2015.

[29] Y. Hashemi and A. H. Banihashemi, "Corrections to 'on characterization of elementary trapping sets of variable-regular LDPC codes'," *IEEE Trans. Inf. Theory*, vol. 61, no. 3, pp. 1508–1508, Mar. 2015.

[30] Y. Hashemi and A. H. Banihashemi, "New characterization and efficient exhaustive search algorithm for leafless elementary trapping sets of variable-regular LDPC codes," *IEEE Trans. Inf. Theory*, vol. 62, no. 12, pp. 6713–6736, Dec. 2016.

[31] Y. Hashemi and A. H. Banihashemi, "An effcient exhaustive search algorithm for elementary trapping sets of variable-regular LDPC codes," in *Proc. Int. Conf. Commun. (ICC)*, Kuala lumpur, Malaysia, May 2016, pp. 6261–6268.

[32] Y. Hashemi and A. H. Banihashemi, "Efficient exhaustive search algorithm for elementary trapping sets of irregular LDPC codes," in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, Barcelona, Spain, Jul. 2016, pp. 2271–2275.

[33] Y. Hashemi and A. H. Banihashemi, "Characterization and efficient exhaustive search algorithm for elementary trapping sets of irregular LDPC codes," submitted to *IEEE Trans. Inf. Theory*, Oct. 2016, available online at: *http://arxiv.org/abs/1611.10014*

[34] Y. Hashemi and A. H. Banihashemi, "Tight lower and upper bounds on the minimum distance of LDPC codes," submitted to *IEEE Comm. Lett.*, May. 2017.

[35] Y. Hashemi and A. H. Banihashemi, "Lower bounds on the size of smallest elementary and non-elementary trapping sets in variable-regular LDPC codes," to appear in *IEEE Comm. Lett.*, May. 2017.

[36] Y. Hashemi and A. H. Banihashemi, "Characterization and efficient search of non-elementary trapping sets with applications to stopping sets," submitted to *IEEE Trans. Inf. Theory*, June 2017.

[37] Y. Hashemi and A. H. Banihashemi, "Characterization and efficient exhaustive search algorithm for elementary trapping sets of irregular LDPC codes," *IEEE Int. Symp. Inform. Theory (ISIT)*, Aachen, Germany, June 2017.

[38] M. Helmling, and S. Scholl, Database of channel codes and ML simulation results. University of Kaiserslautern, 2015. URL: www.uni-kl.de/channel-codes.

[39] M. Hirotomo, M. Mohri, and M. Morii, " A probabilistic computation method for the weight distribution of low-density parity-check codes," in *Proc. IEEE Int. Symp. on Inf. Theory (ISIT)*, Adelaide, SA, USA, Sep. 2005, pp. 4–9.

[40] M. Hirotomo, M. Mohri, and M. Morii, " On the probabilistic computation algorithm for the minimum-size stopping sets of LDPC codes," in *Proc. IEEE Int. Symp. on Inf. Theory (ISIT)*, Toronto, Canada, Jul. 2008, pp. 295–299.

[41] X. Y. Hu, M. P. C. Fossorier, and E. Eleftheriou, " On the computation of the minimum distance of low-density parity-check codes," in *Proc. IEEE Int. Conf. on Commun. (ICC)*, Jun. 2004, pp. 767–771.

[42] X. Y. Hu, and E. Eleftheriou, " A probabilistic subspace approach to the minimal stopping set problem," in *Proc. 4th Int. Symp. Turbo Codes and Related Topics*, Munich, Germany, Apr. 2006, pp. 295–299.

[43] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.

[44] 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.

[45] J. Kang, Q. Huang, S. Lin, and K. Abdel-Ghaffar, "An iterative decoding algorithm with backtracking to lower the error-floors of LDPC codes," *IEEE Trans. Commun.*, vol. 59, no. 1, pp. 64–73, Jan. 2011.

[46] M. Karimi and A. H. Banihashemi, "On characterization of elementary trapping sets of variable-regular LDPC codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 9, pp. 5188–5203, Sep. 2014.

[47] M. Karimi and A. H. Banihashemi, "Efficient algorithm for finding dominant trapping sets of LDPC codes," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp. 6942–6958, Nov. 2012.

[48] M. Karimi and A. H. Banihashemi, "Message-passing algorithms for counting short cycles in a graph," *IEEE Trans. Commun.*, vol. 61, no. 2, pp. 485–495, Feb. 2013.

[49] A. Keha, and T. M. Duman, " Minimum distance computation of LDPC codes using a branch and cut algorithm," *IEEE Trans. Commun.*, vol. 58, no. 4, pp. 1072–1079, Apr. 2010.

[50] S. M. Khatami, L. Danjean, D. V. Nguyen, and B. Vasic, "An efficient exhaustive low-weight codeword search for structured LDPC codes," in *Proc. Inf. Theory and App. Workshop (ITA)*, San Diego, CA, USA, Feb. 2013, pp. 1–10.

[51] S. Khazraie, R. Asvadi and A. H. Banihashemi, "A PEG construction of finite-length LDPC codes with low error floor," *IEEE Comm. Lett.*, vol. 16, pp. 1288 – 1291, Aug. 2012.

[52] K. M. Krishnan and P. Shankar, "Computing the stopping distance of a Tanner graph is NP-hard," *IEEE Trans. Inf. Theory*, vol. 53, no. 6, pp. 2278–2280, June 2007.

[53] G. B. Kyung and C.-C. Wang, "Finding the exhaustive list of small fully absorbing sets and designing the corresponding low error-floor decoder," *IEEE Trans. Commun.*, vol. 60, no. 6, pp. 1487–1498, Jun. 2012.

[54] S. Laendner, O. Milenkovic, and J. B. Huber, "Characterization of small trapping sets in LDPC codes from steiner triple systems," in *Proc. 6th Int. Symp. Turbo Codes and Iterative Inf. Process.*, Brest, France, Sep. 2010, pp. 93–97.

[55] S. Laendner, T. Hehn, O. Milenkovic, and J. B. Huber, "The trapping redundancy of linear block codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 1, pp. 53–63, Jan. 2009.

[56] D. Mackay, R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996.

[57] D. Mackay, "Encyclopedia of sparse graph codes," URL: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[58] [Online] Available: http://www.inference.phy.cam.ac.uk/mackay/codes/.

[59] G. A. Margulis, " Explicit constructions of graphs without short cycles and low density codes," *Combinatorica*, vol. 2, no. 1, pp. 71–78, 1982.

[60] B. D. McKay and A. Piperno, "Practical graph isomorphism, II," *J. Symbolic Computation*, vol. 60, pp. 94–112, 2013.

[61] A. McGregor and O. Milenkovic, "On the hardness of approximating stopping and trapping sets," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1640–1650, Apr. 2010.

[62] O. Milenkovic, E. Soljanin, and P. Whiting, "Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles," *IEEE Trans. Inf. Theory*, vol. 53, no. 1, pp. 39–55, Jan. 2007.

[63] D. V. Nguyen, S. K. Chilappagari, M. W. Marcellin, and B. Vasic, "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.

[64] A. Orlitsky, R. Urbanke, K. Viswanathan, and J. Zhang, " Stopping sets and the girth of Tanner graphs," in *Proc. Int. Symp. on Inf. Theory (ISIT)*, Lausanne, Switzerland, Jun./Jul. 2002, pp. 2.

[65] N. Radford, [Online]. Available: http://www.cs.toronto.edu/ radford/ldpc.software.

[66] T. Richardson, "Error floors of LDPC codes," in *Proc. 41th annual Allerton conf. on commun. control and computing*, Monticello, IL, USA, Oct. 2003, pp. 1426–1435.

[67] T. Richardson, M. A Shokrollahi and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[68] G. Richter, "Finding small stopping sets in the Tanner graphs of LDPC codes," in *Proc. 4th Int. Symp. on Turbo Codes*, Munich, Germany, Apr. 2006, pp. 1–5.

[69] J. Rosenthal, and P.O. Vontobel "Construction of LDPC codes based on Ramanujan graphs and ideas from Margulis," in *Proc. 38th Annu. Allerton Conf. Commun., Control Comput.*, Monticello, IL, USA, Oct. 2000, pp. 248–257.

[70] E. Rosnes and O. Ytrehus, "An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices," *IEEE Trans. Inf. Theory*, vol. 55, no. 9, pp. 4167–4178, Sep. 2009.

[71] E. Rosnes, O. Ytrehus, M. A. Ambroze, and M. Tomlinson, "Addendum to "An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices"," *IEEE Trans. Inf. Theory*, vol. 58, no. 1, pp. 164–171, Jan. 2012.

[72] E. Rosnes, M. A. Ambroze, and M. Tomlinson, "On the minimum/stopping distance of array low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 9, pp. 5204–5214, Sep. 2014.

[73] H. Saeedi and A. H. Banihashemi, "On the design of irregular LDPC code ensembles for BIAWGN channels," *IEEE Trans. Commun.*, vol. 58. no. 5, pp. 1376 – 1382, May 2010.

[74] C. Schlegel and S. Zhang, "On the dynamics of the error floor behavior in (regular) LDPC codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3248–3264, Jul. 2010.

[75] R. Smarandache, and P. O. Vontobel, "Quasi-cyclic LDPC codes: Influence of proto- and Tanner-graph structure on minimum distance upper bounds," *IEEE Trans. Inf. Theory*, vol. 58, no. 2, pp. 585–607, Feb. 2012.

[76] S. Tolouei, "High speed low error floor hardware implementation and fast and accurate error floor estimation of LDPC decoders," Ph.D. dissertation, Dep. of Sys. and Comp. Eng., Carleton University, Ottawa, ON, 2014.

[77] S. Tolouei and A. H. Banihashemi, "Fast and accurate error floor estimation of quantized iterative decoders for variable-regular LDPC codes," *IEEE Comm. Lett.*, vol. 18, pp. 1283–1286, Aug. 2014.

[78] J. Stern, " A method for finding codewords of small weight," in *Coding Theory and Application*, G. Cohen and J. Wolfmann, Eds. New York: Springer-verlag, 1989, pp. 106–113.

[79] R. M. Tanner, " Minimum-distance bounds by graph analysis," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 808–821, Feb. 2001.

[80] R. M. Tanner, T. D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. Int. Symp. Commun. Theory Appl.*, Ambleside, U.K., Jul. 2001, pp. 365–370.

[81] R. M. Tanner, T. D. Sridhara, A. Sridharan, T. Fuja, and D. Costello, "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

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

[83] X. Tao, Y. Li, Y. Liu, and Z. Hu, "On the construction of LDPC codes free of small trapping sets by controlling cycles," to appear in *IEEE Comm. Lett.*, Mar. 2017.

[84] A. Tasdighi, and A. H. Banihashemi, "Efficient search of girth-optimal QC-LDPC codes," *IEEE Trans. Inf. Theory*, vol. 62, no. 4, pp. 1552–1564, Apr. 2016.

[85] A. Tasdighi, and A. H. Banihashemi, "Symmetrical constructions for regular girth-8 QC-LDPC codes," *IEEE Trans. Commun.*, vol. 65, no. 1, pp. 14–22, Jan. 2017.

[86] G. Tzimpragos, C. Kachris, I. Djordjevic, M. Cvijetic, D. Soudris, and I. Tomkos, "A Survey on FEC Codes for 100 G and Beyond Optical Networks," *IEEE Comm. Surveys & Tutor.*, vol. 18, no. 1, pp. 209–221, First Qua. 2016.

[87] A. Vardy, " The intractability of computing the minimum distance of a code," *IEEE Trans. Inf. Theory*, vol. 43, no. 6, pp. 1757–1766, Nov. 1997.

[88] B. Vasic, S. K. Chilappagari, D. V. Nguyen, and S. K. Planjery, "Trapping set ontology," in *Proc. 47th Annu. Allerton Conf. Commun., Control Comput.*, Monticello, IL, USA, Sep. 2009, pp. 1–7.

[89] B. Vasic, and O. Milenkovic, " Combinatorial construction of low-density parity-check codes for iterative decoding," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 1156–1176, Jun. 2004.

[90] C.-C. Wang, S. R. Kulkarni, and H. V. Poor, "Finding all small error-prone substructures in LDPC codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 5, pp. 1976–1999, May 2009.

[91] C.-C. Wang, "On the exhaustion and elimination of trapping sets: Algorithms & the suppressing effect," in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, Nice, France, Jun. 2007, pp. 2271–2275.

[92] H. Xiao, and A. H. Banihashemi, "Estimation of bit and frame error rates of low-density parity-check codes on binary symmetric channels," *IEEE Trans. Commun.*, vol. 55, no. 6, pp. 2234–2239, Dec. 2007.

[93] H. Xiao, and A. H. Banihashemi, "Error rate estimation of low-density parity-check codes on binary symmetric channels using cycle enumeration," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1550–1555, Jun. 2009.

[94] H. Xiao, and A. H. Banihashemi, "Error rate estimation of low-density parity-check codes decoded by quantized soft-decision iterative algorithms," *IEEE Trans. Commun.*, vol. 61, no. 6, pp. 474–483, Feb. 2013.

[95] Y. Zhang, and W. E. Ryan, "Toward low LDPC-code floors: a case study," *IEEE Trans. Commun.*, vol. 57, no. 6, pp. 1566–1573, Jun. 2009.

[96] S. Zhang, and C. Schlegel, "Controlling the error floor in LDPC decoding," *IEEE Trans. Commun.*, vol. 61, no. 9, pp. 3566–3575, Sep. 2013.

[97] X. Zhang, and P. H. Siegel, "Efficient algorithms to find all small error-prone substructures in LDPC codes," in *Proc. Global Telecommun. Conf.*, Houston, TX, USA, Dec. 2011, pp. 1–6.

[98] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. J. Wainwright, "Lowering LDPC error floors by postprocessing," in *Proc. Global Telecommun. Conf.*, New Orleans, LO, USA, Dec. 2008, pp. 3074–3079.

[99] X. Zheng, C. M. Lau, and C. K. Tse, "Constructing short-length irregular LDPC codes with low error floor," *IEEE Trans. Commun.*, vol. 58, no. 10, pp. 2823–2834, Oct. 2010.

[100] *IEEE* standard for information technology local and metropolitan area networks specific requirements part 11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 5: enhancements for higher throughput, *IEEE* Std. 802.11n-2009, Oct. 29, 2009.

[101] *IEEE* standard for local and metropolitan area networks Part 16: air interface for fixed and mobile broadband wireless access systems amendment 2: physical and medium access control layers for combined fixed and mobile operation in licensed bands and corrigendum 1, *IEEE* Std. 802.16e-2005 and 802.16-2004/Cor 1-2005, Feb. 28, 2006.

[102] *IEEE* P802.22 Draft Standard for Wireless Regional Area Networks Part 22: Cognitive Wireless RAN Medium Access Control (MAC) and Physical Layer (PHY) Specification: Policies and Procedures for Operation in the TV Bands, *IEEE P802.22 Draft Standard*, D0.3, May 2007.

[103] *IEEE* Standard for Information Technology-Telecommunications and Information Exchange between Systems-Local and Metropolitan Area Networks-Specific Requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specification, *IEEE Std. 802.3an*, Sep. 2006.

[104] *ETSI* Standard TS 102 005: Digital Video Broadcasting (DVB): frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2), ETSI Std. TS 102 005, Mar. 2010.

[105] *ETSI* Standard TR 102 376 V1.1.1: Digital Video Broadcasting (DVB) User guidelines for the second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVBS2), ETSI Std. TR 102 376, Feb. 2005.

[106] TM Synchronization and Channel Coding. Reston, VA, Consultative Committee for Space Data Systems, Recommended Standard 131.0-B-2, Aug. 2011 [Online]. Available: http://public.ccsds.org/publications/archive/131x0b2ec1.pdf