An Integer Programming-Based Search Technique for Error-Prone Structures of LDPC Codes

Abdullah Sarıduman[a], Ali E. Pusane[a], Z. Caner Taşkın[b]

[a] *Department of Electrical and Electronics Engineering, Boğaziçi University, Istanbul 34342, Turkey*
[b] *Department of Industrial Engineering, Boğaziçi University, Istanbul 34342, Turkey*

Abstract

In this paper, an efficient, general framework is presented for finding common, devastating error-prone structures (EPS) of any finite-length low-density parity-check (LDPC) code. The smallest stopping set for the binary erasure channel (BEC), the smallest fully absorbing set, the smallest absorbing set, and the smallest elementary trapping set for the binary symmetric channel (BSC) are found and the dominant EPS are enumerated. The method involves integer programming optimization techniques, which guarantees that the results are provably optimal.

Keywords: Trapping sets, stopping sets, absorbing sets, fully absorbing sets, elementary absorbing sets, integer programming.

1. Introduction

Low-density parity-check (LDPC) codes, developed by Gallager in 1962, have become one of today's most popular error control techniques due to their capacity-approaching error performance [1]. A well constructed LDPC code has a bipartite graph representation, called the Tanner graph, which does not have small error-prone structures (EPS); a property needed to ensure that the code has high error floor performance. The error floor is an abrupt degradation in the frame error rate performance in the high signal-to-noise ratio region that arises because of the presence of EPS in the code's graph representation and low weight codewords. For many LDPC code families, EPS are more dominant than the low weight codewords in the error floor region, as in the example of [2], since there are EPS with fewer sizes than the weight of the minimum codeword weight. Although all of the EPS appear as a result of the existence of (short) cycles in the graph representation, their detailed structure is determined by the channel used. For instance, stopping sets are known to be the main problematic structures for communication over the binary erasure channel (BEC), whereas trapping sets are the main problematic structures for communication over the binary symmetric channel (BSC), as well as the additive white Gaussian noise channel (AWGNC) [3], [4]. Among the family of trapping sets, it is demonstrated that elementary trapping sets, absorbing sets, and fully absorbing sets are the main structures that determine the error floor performance of an LDPC code [4], [5], [6]. It was also shown that problematic sets of smaller sizes are dominant in the error floor region, since the probability of occurrence is higher for these structures [4].

A very important code property of LDPC codes is the minimum distance, i.e., the smallest Ham-

This work was supported in part by the Scientific and Technological Research Council of Turkey (TÜBİTAK) (Grant no. 113M499) and European Union FP7 Marie Curie International Reintegration Grant (Grant no. 268264).

ming distance between any two distinct codewords of a code. The knowledge of the minimum distance and the small dominant problematic structures of an LDPC code allows for approximating and/or bounding its performance for maximum-likelihood decoding and iterative decoding, respectively. For example, in [4], Richardson uses trapping sets to estimate the error floor performance of LDPC codes. Moreover, this knowledge provides the code designer with a metric to further tweak the code design to achieve better error performance. For instance, [7] proposed an efficient method to improve LDPC code designs in terms of the error floor performance when small dominant problematic structures are known at the decoder. Therefore, a general framework to find dominant problematic structures and calculate the minimum distance of LDPC codes is vital.

There are several studies present in the literature that calculate the minimum distance of an LDPC code. Although the calculation of the minimum distance is an NP-hard problem, a very efficient method is proposed in [8] to calculate the minimum distance of an LDPC code using the branch-and-cut algorithm. Its idea mainly inspired our work. In [8], integer programming (IP) is employed to model the code constraints and calculate the minimum distance. Similar to the minimum distance problem, finding most dominant problematic structures, minimum trapping sets and minimum stopping sets, are not easy tasks; in fact, these problems are also shown to be NP-hard in [9, 10].

Due to the complexity of finding minimum error prone structures, several studies in the literature instead focus on finding an inexhaustive list of EPS. Dolecek *et al.* in [11] designed an FPGA simulation to generate a partial list of EPS and estimate the error floor. Others have also considered dominant EPS by utilizing importance sampling and the error impulse methods in [12, 13]. However, finding the small EPS size and obtaining an inexhaustive EPS list are not comparable tasks, since the latter one does not guarantee to include the smallest set. There are also some studies in the literature that aim to calculate the smallest EPS size, including [14]. However, in that study, the authors assume that an extensive amount of prior knowledge, namely an exhaustive list of the cycles, is given as an input to the algorithm. This makes a fair comparison between their algorithm and the one proposed in this paper impossible. Another similar study in the literature is [15], where the smallest EPS and an exhaustive list of EPS are obtained. The main difference is in the calculation of the absorbing sets. While the authors in [15] first enumerate stopping sets and then construct other EPS based on these stopping sets, in our approach, we devise new IP models for the other EPS types as well.

In this paper, we propose an efficient, general framework to find the smallest dominant problematic structures and enumerate small dominant problematic structures of an LDPC code by means of the IP optimization technique. For completeness, we also review the IP model developed in [8]. In [4] and [5], it was shown that the problematic structures for communication over the BSC also cause problems for communication over the AWGNC. Therefore, determining these small structures can be beneficial for the AWGNC as well as the BSC. Our proposed method takes the parity-check matrix representation of an LDPC code as an input and finds the important parameters of an LDPC code, including the smallest absorbing set size, the smallest fully absorbing set size, the smallest elementary absorbing set size, and the smallest stopping set size. An iterative IP algorithm is then proposed to enumerate all EPS. The solution is provably optimal due to the use of IP, i.e., if the proposed algorithm succeeds, the output is guaranteed to be optimal. The proposed algorithm is tested for LDPC codes with lengths suitable for practical implementations and it is demonstrated that the algorithm executes in a reasonable amount of time for on modern computers.

The rest of the paper is organized as follows: Section 2 contains basic relevant definition and notations. Section 3 describes the proposed integer programming models for finding small EPS. Section 4 describes an algorithm to enumerate all EPS of a given code and Section 5 provides some numerical results. Finally, the paper is concluded in Section 6.

2. Preliminaries

2.1. Bipartite graph representation

An (n, k) LDPC code can be represented in terms of a bipartite graph, G (also called the Tanner graph), with two sets of nodes: n variable nodes, $V = \{v_1, \ldots, v_n\}$, and $m = n - k$ check nodes, $C = \{c_1, \ldots, c_m\}$ [16]. Variable nodes correspond to code symbols and check nodes correspond to parity-check equations. An edge connects a variable node to a check node if and only if the corresponding code symbol participates in the corresponding parity-check equation. The nodes connected to the i-th variable (j-th check) node are referred to as its neighbors and denoted as $N(v_i)$ $(N(c_j))$. The degree of a node, therefore, is given as $|N(v_i)|$ or $|N(c_j)|$. The induced subgraph of subset $x \subseteq V$, G_x, is a bipartite graph containing x, $N(x)$ and their corresponding edges. $E(x)$ represents the set of the check nodes in $N(x)$ with even degree in G_x and $O(x)$ represents the set of check nodes in $N(x)$ with odd degree in G_x. The adjacency matrix of the Tanner graph, $\mathbf{H}_{m\times n}$, is an $m \times n$ matrix that satisfies $\mathbf{vH^T} = 0 \mod (2)$ if and only if $\mathbf{v} = \{v_1, v_2, \ldots, v_n\}$ is a codeword in the associated code.

Example 1. *The Tanner graph of an (8,3) LDPC code with parity-check matrix* $\mathbf{H}$ *is represented in Figure 1. The filled circles represent the variable nodes and the empty squares represent the check nodes. The corresponding parity-check matrix is given by*

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \qquad (1)$$

2.2. Problematic structures

2.2.1. BEC

For communications over the BEC, the received vector consists of both correctly received and erased symbols. In order to eliminate these erased symbols, the erasure decoder iteratively determines their values, utilizing check nodes with single erased neighbors in each iteration.

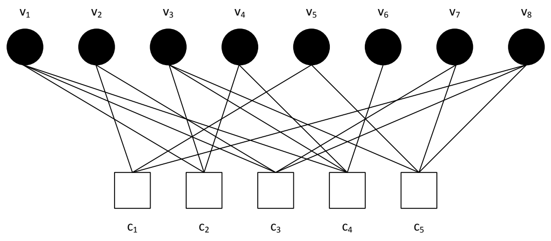


Figure 1: The Tanner graph of an (8,3) LDPC code.

A decoding error occurs when there are still erased symbols in the received vector and all check nodes have either zero or at least two erased symbols in their neighborhoods. The decoder gets stuck on such a configuration of erased symbols. Note that this configuration can appear directly at the output of the channel or at any time during the decoding operations. It is therefore crucial to identify these configurations, namely stopping sets (SS), and study their structures. The following is an example of a small SS.

Example 2. *For the (8,3) LDPC code, whose Tanner graph is given in Figure 1,* $x = \{v_2, v_5, v_7\}$ *is a SS, since the degrees of check nodes* c_1*,* c_3*, and* c_5 *are all equal to two in* G_x*.*

2.2.2. BSC

For communication over the BSC, the received vector consists of correctly and erroneously received symbols and it is the decoder's task to determine which symbols are in error. The most popular decoding algorithm to be used over the BSC is Gallager's bit-flipping algorithm. The bit-flipping algorithm, as its name suggests, flips the value of a variable node when at least b of its neighboring check nodes are unsatisfied. Such a flip would then reduce the number of unsatisfied check nodes in the Tanner graph. The decoder continues flipping the variable nodes until a maximum number of iterations is reached or there are no remaining unsatisfied check nodes.

A decoding error occurs when there are still unsatisfied check nodes in the graph, yet the decoder fails to flip any more variable nodes. This occurs when each variable node in the graph has more satisfied neighboring check nodes than unsatisfied ones. Error configurations that give rise to such a result are therefore of interest to us.

We begin by defining the structure of trapping sets, the general framework of EPS in the Tanner graph.

Definition 3. *$x \subseteq V$ is an (s,t) trapping set if $|x| = s$ and $|O(x)| = t$.*

An (s,t) trapping set with small values for both of the parameters has the potential to harm the decoding performance, since a small value of s makes it more likely to observe such an error structure at the channel output and a small value of t makes it more likely for a decoder to get stuck during the decoding iterations. A special class of trapping sets are called elementary trapping sets and are defined as:

Definition 4. *An (s,t) elementary trapping set x is an (s,t) trapping set such that $|N(c_j)| = 1$, $\forall c_i \in O(x)$ and $|N(c_j)| = 2$, $\forall c_j \in E(x)$.*

Although trapping sets and elementary trapping sets vaguely describe the potential damage they could cause to the decoding process, they are mostly conceptual structures and their damage is heavily dependent on the chosen value of the (s,t) pair. A subclass of them, called absorbing sets, provide us with a more realistic definition. An absorbing set (AS) is a trapping set such that all the variable nodes that take part in the AS are connected to more satisfied check nodes than unsatisfied ones. This means that an AS, on itself, would be able to stop the decoding process. In fact, for the Gallager-B variant of the bit-flipping algorithm with invariant threshold $b{=}(|N(v_i)|-1)/2$, all of the trapping sets are exactly AS.

An AS is called fully absorbing, when all variable nodes in the graph, both within the induced graph and elsewhere, have more satisfied neighboring check nodes than unsatisfied ones. Finally, similar to the trapping set definitions, an AS is called elementary if the induced graph only has check nodes of weights 1 and 2. The following two examples demonstrate the conditions for the decoder to stop processing when an AS is encountered.

Example 5. *$x = \{v_1, v_3\}$ is an AS for the Tanner graph in Figure 1, since both v_1 and v_3 have less odd degree neighbors in their induced subgraph, c_3 or c_5, than their remaining neighbors, c_2 and c_4. On the one hand, it is not a fully absorbing set (FAS), because v_8 has two odd neighbors, c_3 and c_5, from G_x and only one neighbor, c_1, from G'_x. On the other hand, it is an elementary absorbing set (EAS) owning to fact that every check node in G_x has degree two.*

Example 6. *$x = \{v_1, v_3, v_8\}$ is an FAS, since it is an AS and every variable node in G'_x has less neighbors from $O(x)$ than their remaining neighbors.*

3. Integer Programming Formulations for Finding Small EPS

Linear programming (LP) is a very popular optimization technique that is widely used for modeling and solving optimization problems encountered in many different areas. In the LP method, every decision point in the problem is modeled as a *decision variable*. *Constraints* satisfied by the feasible solutions are expressed as linear functions of the decision variables. Similarly, the *objective function* is defined as a linear function of the decision variables and measures the quality of feasible solutions. A standard form of an LP model is given as

$$\text{minimize } \mathbf{c}^T\mathbf{x} \tag{2}$$
$$\text{such that: } \mathbf{A}\mathbf{x} = \mathbf{b} \tag{3}$$
$$\mathbf{D}\mathbf{x} \geq \mathbf{e} \tag{4}$$
$$\mathbf{F}\mathbf{x} \leq \mathbf{g} \tag{5}$$
$$\mathbf{x} \geq 0, \tag{6}$$

where $\mathbf{x}$ represents the vector of decision variables, $\mathbf{A}$, $\mathbf{D}$, $\mathbf{F}$ are matrices that represent constraint coefficients, $\mathbf{c}$ is a vector representing objective function coefficients and $\mathbf{b}$, $\mathbf{e}$, $\mathbf{g}$ are vectors representing constraint right-hand-side values. An *optimal solution* that is guaranteed to minimize the objective function can be found in polynomial time and all of the constraints are simultaneously satisfied [17]. There are various algorithms for solving LP problems. Some interior point algorithms such as Karmarkar's algorithm and ellipsoid method have polynomial time complexity. While the worst case complexity of the simplex algorithm is exponential, its works very well in practice, and is one of the most popular methods for solving LPs [18].

LP assumes that the decision variables can take non-integer values. On the other hand, this assumption is not realistic in some applications and the decision variables must take integer values. The form of LP, which assumes that all of the decision variables must take on integer values, is called IP. A standard form of an IP model is then given as a modification of the LP model, where equation (6) is replaced with

$$\mathbf{x} \in \mathbb{Z}. \tag{7}$$

Although LP problems can be solved in polynomial time, the solution of IP problems is NP-hard. However, in practice, IP problems can be efficiently solved for quite large problem sizes using algorithms such as branch-and-bound and branch-and-cut [19].

To solve an IP problem, such as the ones that are of interest to us in this paper, the problem is first transformed into an LP problem via *LP relaxation*. In this relaxed model, integrality restrictions on the decision variables are removed to allow for fractional values. This modification significantly simplifies the solution and an optimal solution of the LP relaxation can be found in polynomial time as discussed earlier. Interested readers can refer to [17, 18, 20] for more details. Although very easy to find, the optimal objective function value of the relaxed LP problem is not necessarily equal to that of the original IP problem. However, it can be used to obtain a valid lower bound (upper bound) for the optimal objective function value of the minimization (maximization) problem at hand. Therefore, a branch-and-bound algorithm can be used to solve the original IP problem. In this paper, we employ the branch-and-bound algorithm given in Algorithm 1 to solve the minimization problems of interest.

Although branch-and-bound algorithms are very effective in solving the IP problems at hand, the number of internal nodes that the algorithm needs to traverse can increase very fast, resulting in a situation where an optimum solution cannot be found in a reasonable amount of time. In order to improve on

Algorithm 1 Branch-and-bound algorithm

Step 1: Set the active node count $n = 1$ (LP relaxation of the original problem is an active node). Set the candidate objective value $z = \infty$.

Step 2: If there is no active node, $n = 0$, then the candidate solution is the optimal solution. Else, go to Step 3.

Step 3: Solve the LP relaxation of any active node and set this node as the current node. If there is no feasible solution of the current node or the optimal objective function value of the current node, z^*, is larger than the candidate objective function value z, go to Step 4. Else, if the solution has fractional values, then go to Step 5. Otherwise, go to Step 6.

Step 4: Process the current node. Decrease the active node number by 1. Go to Step 2.

Step 5: Branch on the current node. Select a fractional variable, x_i. Let x_i be between y and $y + 1$, where $y \in \mathbb{Z}$. Generate child nodes by adding new constraints as $x_i \leq y$ and $x_i \geq y + 1$. Increase the active node number, n, by 1 and go to Step 2.

Step 6: Set the current node as the candidate solution. Set the the candidate objective value as $z = z^*$. Decrease n by 1. Go to Step 2.

this situation and reduce the number of active nodes, branch-and-cut algorithms are employed. These algorithms produce additional inequalities, called *cuts*, that are satisfied by all integer solutions, but not necessarily by fractional solutions. Once the algorithm produces a *violated cut*, i.e., an inequality that is not satisfied by the current fractional optimum solution of the relaxed LP problem, the cut is added to the LP relaxation, which is then re-solved. Branching is performed on the corresponding node only if a violated cut cannot be generated. Strong cuts that manage to eliminate the current best fractional best solution significantly improve solvability of the IP problem. In this paper we utilize CPLEX 12.4 software, which employs the simplex, branch-and-bound, and branch-and-cut algorithms to efficiently solve IP problems.

In this section, we develop IP models to search for dominant problematic error sets for (n, k) LDPC codes with associated parity-check matrices $\mathbf{H}_{m\times n}$ over the BEC and BSC. In all of our IP models, the decision variables represent the variable nodes and it is assumed that a decision variable takes on the value 1 if the corresponding variable node is in error (BSC) or is erased (BEC).

3.1. Finding the Minimum Distance

Minimum distance computation of an LDPC code via IP was achieved in [8]. The authors utilized the property of all linear codes that the summation of any two codewords is also a codeword and therefore, the minimum distance of a linear block code is the minimum Hamming weight of its non-zero codewords. Their IP model finds a codeword having minimum Hamming weight. The IP formulation that is used to find a codeword $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ with minimum Hamming weight was modeled in [8] as:

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{8}$$

$$\text{s.t.:} \quad \sum_{j=1}^{n} h_{i,j} x_j = 2k_i, \quad i = 1, 2, \ldots, m \tag{9}$$

$$\sum_{j=1}^{n} x_j \geq 1 \tag{10}$$

$$k_i \in \mathbb{Z}, \quad i = 1, 2, \ldots, m \tag{11}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n. \tag{12}$$

The objective function in this model is the sum of the components of $\mathbf{x}$, i.e., the Hamming weight of $\mathbf{x}$. The constraint set (9) forces $\mathbf{x}$ to satisfy $\mathbf{H}\mathbf{x}^{\mathrm{T}} = 0 \mod (2)$, i.e., $\mathbf{x}$ is a codeword. The constraint set (10) further forces $\mathbf{x}$ to be non-zero. Finally, variable k is used to linearize and represent the $\mod (2)$ operation.

3.2. Finding the Minimum Stopping Set

We start with giving a formal definition of stopping sets.

Definition 7. *$x \subseteq V$ is a SS if $|N(c_j) \cap G_x| \geqslant 2$, $\forall c_j \in N(x)$.*

This defines a subset x of the variable nodes (used to determine the erased nodes) and requires that the connected check nodes (in the induced graph) have multiple connections to x. Using this notation, an extension of the model given in (8)-(12) is obtained to find the minimum SS size over the BEC:

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{13}$$

$$\text{s.t.: } 2u_i \le \sum_{j=1}^{n} h_{i,j} x_j \quad i = 1, 2, \ldots, m \tag{14}$$

$$\sum_{j=1}^{n} h_{i,j} x_j \le \sum_{j=1}^{n} h_{i,j} u_i, \quad i = 1, 2, \ldots, m \tag{15}$$

$$\sum_{i=1}^{n} x_i \ge 1 \tag{16}$$

$$u_i \in \{0, 1\}, \quad i = 1, 2, \ldots, m \tag{17}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n. \tag{18}$$

A decision variable, x_j, is equal to 1 if and only if the corresponding variable node is a member of the stopping set. Therefore, in order to find the minimum SS, the objective function given in (13) is simply defined as the summation of the decision variables. Modeling what constitutes a stopping set is a little trickier than modeling a codeword, though. We start with the observation that a check node in the existence of a stopping set has degree either zero (check node outside the induced subgraph) or at least two (check node inside the induced subgraph). To be able to model these two conditions, we first calculate the degree of a check node c_i, $i = 1, 2, \ldots, m$, as $d(c_i) = \sum_{j=1}^{n} h_{i,j} x_j$. In (14), we define a binary variable u_i that determines whether check node c_i is in the induced subgraph: u_i takes on the value 1 if c_i is in the induced subgraph and the value 0, otherwise. If the check node is in the induced subgraph ($u_i = 1$), the constraint (14) guarantees that the degree $d(c_i)$ is at least two and the constraint (15) becomes inactive, since

$$\sum_{j=1}^{n} h_{i,j} x_j \le \sum_{j=1}^{n} h_{i,j} \tag{19}$$

is satisfied for every possible binary $\mathbf{x}$ vector. If the check node is not in the induced subgraph ($u_i = 0$), the constraint (15) guarantees that $\sum_{j=1}^{n} h_{i,j} x_j$ is zero. Hence, (14) and (15) guarantee that the degree $d(c_i)$ is either zero or at least two for all $i = 1, 2, \ldots, m$. Finally, the constraint (16) is necessary to make the solution $x_j = 0$, $j = 1, 2, \ldots, n$, an infeasible choice.

When this IP model is employed, it computes the best feasible solution by using the branch-and-cut algorithm. Since our constraints ensure that this solution is a stopping set, the minimum set size found by IP model is guaranteed to be the size of the minimum stopping set.

3.3. Finding the Minimum Fully Absorbing Set

Although fully absorbing sets are subsets of absorbing sets, they are easier to define using an IP model, and therefore, we start with giving a formal definition of them.

Definition 8. *An (s, t) absorbing set x is an FAS if $|N(v_i) \setminus O(x)| > |N(v_i) \cap O(x)|$, $\forall v_i \in V$.*

The vector x in this definition determines the positions of errors and can be observed at either the channel output or at some point during the iterations.

An IP model to find the minimum FAS size over the BSC can be given as:

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{20}$$

$$\text{s.t.: } \sum_{j=1}^{n} h_{i,j} x_j + s_i = 2k_i, \; i = 1, 2, \ldots, m \tag{21}$$

$$\sum_{i=1}^{m} h_{i,j} s_i \le \sum_{i=1}^{m} h_{i,j}/2, \quad j = 1, 2, \ldots, n \tag{22}$$

$$\sum_{i=1}^{m} s_i \ge 1 \tag{23}$$

$$s_i \in \{0, 1\}, \; k_i \in \mathbb{Z}, \; i = 1, 2, \ldots, m \tag{24}$$

$$x_j \in \{0, 1\}, \; j = 1, 2, \ldots, n. \tag{25}$$

By definition, an error set is a FAS if and only if every variable node in the graph has more even-degree neighbors than odd-degree neighbors. This simple condition is sufficient to determine whether an error set is a FAS. We define a binary variable s_i to represent the situation of the check node c_i: s_i takes on the value 1 is c_i is unsatisfied and the value 0, otherwise. These values are set via the constraint set (21). The constraint set (22) guarantees that the number of odd-degree neighbors of a variable node v_j is less than half of its degree, i.e., v_j has mode even-degree neighbors than odd-degree neighbors. Finally, the constraint (23) is necessary to make the solution $x_j = 0$, $j = 1, 2, \ldots, n$, an infeasible choice.

3.4. Finding the Minimum Absorbing Set

Having formally defined the FAS, the AS definition becomes much easier:

Definition 9. *An (s,t) AS x is an (s,t) trapping set such that $|N(v_i) \setminus O(x)| > |N(v_i) \cap O(x)|$, $\forall v_i \in x$.*

An IP model to find the minimum AS size over the BSC can be given as:

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{26}$$

$$\text{s.t.:} \quad \sum_{j=1}^{n} h_{i,j} x_j + s_i = 2k_i, \; i = 1, 2, \ldots, m \tag{27}$$

$$\sum_{i=1}^{m} h_{i,j} s_i \leq \sum_{i=1}^{m} h_{i,j}(1 - x_j/2), \quad j = 1, 2, \ldots, n \tag{28}$$

$$\sum_{i=1}^{m} s_i \geq 1 \tag{29}$$

$$s_i \in \{0,1\}, \; k_i \in \mathbb{Z}^+, \; i = 1, 2, \ldots, m \tag{30}$$

$$x_j \in \{0,1\}, \; j = 1, 2, \ldots, n. \tag{31}$$

Finding the minimum AS size is very similar to finding the minimum FAS size. The only difference is that, for the minimum AS size, we must consider only the variable nodes in the induced subgraph rather than the entire graph. Therefore, the constraint set (22) should be valid for the values of $j = 1, 2, \ldots, n$ for which $x_j = 1$. We achieve this by replacing (22) by (28). If x_j is zero, then the constraint (28) becomes

$$\sum_{i=1}^{m} h_{i,j} s_i \leq \sum_{i=1}^{m} h_{i,j}, \tag{32}$$

which is satisfied for all possible $\mathbf{s}$ vectors and therefore has no effect. If x_j is one, however, the constraint becomes identical to (22).

3.5. Finding the Minimum Elementary Absorbing Set

Finally, an EAS is also defined as an extension of AS.

Definition 10. *A (a,b) absorbing set x is an EAS if $|N(c_j)| = 1$, $\forall c_j \in O(x)$ and $|N(c_j)| = 2$, $\forall c_j \in E(x)$.*

An IP model to find the minimum EAS size over the BSC can be given as:

$$\text{minimize} \sum_{j=1}^{n} x_j \tag{33}$$

$$\text{s.t.:} \quad \sum_{j=1}^{n} h_{i,j} x_j + s_i = 2k_i, \; i = 1, 2, \ldots, m \tag{34}$$

$$\sum_{i=1}^{m} h_{i,j} s_i \leq \sum_{i=1}^{m} h_{i,j}(1 - x_j/2), \quad j = 1, 2, \ldots, n \tag{35}$$

$$\sum_{i=1}^{m} s_i \geq 1 \tag{36}$$

$$s_i, k_i \in \{0,1\}, \; i = 1, 2, \ldots, m \tag{37}$$

$$x_j \in \{0,1\}, \; j = 1, 2, \ldots, n. \tag{38}$$

To be an EAS, the induced subgraph of an AS must have only check nodes with degree one or two. Therefore, the degree of a check node in the induced subgraph, $d(c_i) = \sum_{j=1}^{n} h_{i,j} x_j$, cannot be greater than two for the check nodes in the induced subgraph. This condition is achieved by placing the constraint set (37), where the restriction on the value of k_i guarantees that the check node degrees cannot exceed two.

4. Enumerating All EPS

The IP models developed in Section 3 for finding small error-prone substructure sizes are useful tools to determine the error correction potential of a given code. However, in some applications, we further would like to list and enumerate all of these structures rather than knowing their sizes. Enumeration algorithms, such as the one proposed in this section, help the code designer tweak his code and/or decoder design. The proposed enumeration algorithm, presented in Algorithm 2, is quite generic and can be employed in conjunction with all of the IP models we have discussed so far.

Algorithm 2 Enumerating all EPS of size less than t

```
1: Input: Parity-check matrix $\mathbf{H}$ of the LDPC code, maximum size $t$, IP models
2: $size \leftarrow 0$
3: while
4:     Solve the IP; obtain the solution vector as $\mathbf{x}^*$.
5:     if $w_{\mathrm{H}}(\mathbf{x}^*) \geq t$, where $w_{\mathrm{H}}(.)$ is the Hamming weight of a vector,
6:         break while
7:     end
8:     if $w_{\mathrm{H}}(\mathbf{x}^*) > size$
9:         Restore the original IP model
10:        Add $\sum_{j=1}^{n} x_j \geq w_{\mathrm{H}}(\mathbf{x}^*)$ as a new constraint to the IP
11:    end
12:    Add $\mathbf{x}^*$ to the solution table
13:    Add $\sum\limits_{j:x_j^*=0} x_j + \sum\limits_{j:x_j^*=1} (1-x_j) \geq 1$ as a new constraint to the IP model
14:    Set $size = w_{\mathrm{H}}(\mathbf{x}^*)$
15:end
16:Output: The solution table which presents the all EPS of size less than $t$
```

The algorithm starts by solving the original IP problem to find the minimum error-prone substructure and its size. Once a solution is obtained, its size is compared to the maximum search size t in Steps 5-7 and the algorithm is stopped if it exceeds t. The list of EPS it has found so far is given as output. Otherwise, if the size is smaller than t, then the obtained solution $\mathbf{x}^*$ is added to the list and a new constraint is added to the IP model to ensure that the same solution cannot be reached in subsequent runs (Steps 12 and 13). We also set the variable *size* to the substructure size to store this value in the subsequent runs (Step 14). An enumeration of all EPS is possible using this model, however, the number of additional constraints added to the model may be too large for large values of t. To overcome this problem and obtain a more efficient algorithm, Steps 8-11 are added for model simplification. In effect, Step 8 compares the size of the newest solution to the stored value. If the size has increased, there is no need to use the additional constraints added in Step 13 to make every one of the past solutions infeasible; we can simply restore the original IP model and add a single constraint to eliminate all solutions with size smaller than the current one. This allows us to avoid adding too many constraints to which would result in reduced performance. The following example demonstrates the complexity savings achieved by these steps.

Example 11. *The smallest AS for the quasi-cyclic Tanner code of length $n = 155$ has size 4 and there are 465 distinct such substructures in the code's graph representation. Therefore, in order to find an AS of size 5, we would need some 465 additional constraints in our IP model, whereas a single constraint $\sum_{j=1}^{n} x_j \geq 6$ would easily replace these constraints.*

5. Numerical Results

In this section, we present dominant error-prone substructure search results for several different classes of LDPC codes of practical lengths. In order to verify the applicability of the proposed optimization algorithms, we consider both a family of circulant-based quasi-cyclic LDPC codes, namely Tanner codes [21], and families of randomly constructed LDPC codes, both fully random permutation matrix-based codes and structurally random codes obtained by the progressive edge growth (PEG) algorithm [22]. LDPC codes obtained by using the

		d_{min}						SS					
Code Structure	Size	Optimal	LB	UB	Processed Nodes	Active Nodes	Time	Optimal	LB	UB	Processed Nodes	Active Nodes	Time
	55	6			78	0	0	6			193	0	0
	155	20			823172	0	501	18			153633	0	194
	310	20			1160588	0	774	18			330905	0	484
	620	20			1405230	0	1215	18			580591	0	1019
Algebraic (Tanner Codes)	775	24			751388	0	3723	time	20	24	861251	76059	10800
	930	24			1232906	0	8553	time	19	24	877041	209212	10800
	1085	24			818594	0	6538	time	20	24	842984	111764	10800
	2170	memory	22	24	1907336	25753	6485	time	10	24	428609	283900	10800
	3255	time	19	24	1812888	359272	10800	memory	10	24	668438	445921	1205
	4340	time	19	24	1975515	482390	10800	time	10	24	879654	596281	10800
	8680	memory	7	24	512074	357411	2047	memory	5	24	392227	310930	2516
	54	4			92	0	0	4			271	0	0
	156	8			4556	0	0	8			8767	0	2
	312	14			210928	0	70	11			129097	0	52
	504	12			471828	0	447	11			106388	0	75
Random (PEG)	600	16			192098	0	184	16			557244	0	2676
	756	memory	9	759	1916443	1735067	2391	time	12	25	903124	300790	10800
	1008	memory	8	1008	1796164	1674921	12917	time	10	58	1428634	1147657	10800
	2400	memory	5	2400	13117244	1235735	478	memory	6	196	12409636	1220292	921
	3600	memory	4	3600	960896	883357	576	memory	4	192	886795	882320	874
	4200	memory	3	4200	867309	784259	632	memory	3	347	793994	789226	989
	8010	memory	1	8010	522569	481099	656	memory	1	634	478398	475061	1053
	54	2			0	0	0	2			0	0	2
	156	4			19	0	0	4			671	0	0
	312	2			281	0	0	2			1658	0	0
	504	2			35	0	0	2			98	0	0
Random (Permutation)	756	8			89	0	0	8			179	0	0
	810	8			572327	0	1015	8			321340	0	872
	900	2			102	0	0	2			184	0	0
	1008	memory	9	336	2131109	1947481	2846	time	11	33	1293694	6309684	10800
	2400	memory	5	800	1307781	1228845	585	memory	5	178	1250538	1202679	990
	3600	memory	4	1200	950432	874655	545	memory	4	269	881307	874775	848
	4200	memory	4	1400	862352	781552	666	memory	4	292	792998	788318	10502
	8010	memory	1	2670	517598	476433	663	memory	1	625	478825	476433	1093

Table 1: Dominant minimum problematic structures of LDPC codes

		FAS				AS				EAS			
Code Structure	Size	Optimal	Processed Nodes	Active Nodes	Time	Optimal	Processed Nodes	Active Nodes	Time	Optimal	Processed Nodes	Active Nodes	Time
	55	4	103	0	0	3	50	0	0	3	33	0	0
	155	5	9938	0	1	4	7125	0	0	4	3198	0	0
	310	5	16608	0	3	4	8801	0	1	4	6091	0	1
	620	5	36706	0	9	4	16283	0	8	4	8076	0	3
Algebraic (Tanner Codes)	775	4	14305	0	4	4	22430	0	14	4	13390	0	14
	930	4	15834	0	6	4	22617	0	19	4	12964	0	18
	1085	4	18081	0	7	4	27258	0	34	4	18054	0	36
	2170	4	33750	0	29	4	46559	0	115	4	24391	0	147
	3255	4	48988	0	66	4	71368	0	335	4	29929	0	343
	4340	4	64553	0	118	4	66001	0	1655	4	41187	0	605
	8680	4	229237	0	311	4	2290237	0	2652	4	71904	0	2994
	54	3	232	0	0	3	456	0	0	3	255	0	0
	156	3	847	0	0	3	2647	0	0	3	1643	0	0
	312	3	1792	0	0	3	5214	0	1	3	3687	0	4
	504	3	1784	0	0	3	5410	0	1	3	3459	0	0
Random (PEG)	600	3	3260	0	1	3	9350	0	5	3	6050	0	4
	756	3	4005	0	2	3	12060	0	13	3	8544	0	7
	1008	3	5827	0	3	3	15359	0	21	3	9797	0	17
	2400	3	14798	0	21	3	12351	0	77	3	15197	0	164
	3600	3	20064	0	35	3	17592	0	302	3	17801	0	425
	4200	3	23670	0	47	3	24316	0	264	3	19185	0	697
	8010	3	44579	0	133	3	48779	0	1229	3	26591	0	26
	54	2	62	0	0	2	106	0	0	2	96	0	0
	156	2	194	0	0	2	711	0	0	2	755	0	0
	312	2	185	0	0	2	677	0	1	2	722	0	1
	504	2	398	0	0	2	1159	0	0	2	1223	0	0
Random (Permutation)	756	2	980	0	0	2	2628	0	2	2	2377	0	3
	810	2	1057	0	0	2	2589	0	4	2	2100	0	5
	900	2	1346	0	0	2	2414	0	6	2	2536	0	2
	1008	2	1354	0	2	2	2467	0	7	2	2641	0	7
	2400	2	2495	0	8	2	5811	0	108	2	5076	0	126
	3600	2	4207	0	13	2	8864	0	359	2	8291	0	660
	4200	2	5758	0	18	2	9870	0	866	2	9478	0	866
	8010	2	11101	0	78	2	18774	0	9586	2	14544	0	4132

Table 2: Dominant minimum problematic structures of LDPC codes

PEG algorithm have the randomness property of the underlying code as well as maximal girth values, a property that manifests itself in increased values for the smallest error-prone substructure sizes. The PEG construction relies on establishing edges to Tanner graph of the LDPC codes in a way that avoids small cycles. The selection of an edge is determined by the potential impact on the girth. After the Tanner graph is updated by establishing best-choice edge, the placement procedure is repeated for the remaining edges. Randomly constructed LDPC codes, because of the lack of the algebraic structure, have some problems such as encoding data and analyzing code performance. Algebraically constructed LDPC codes can deal with these issues, however, the girth, minimum distance, and error performance of algebraically constructed LDPC codes are not good as random codes. In the recent literature, Tanner codes, comprised of blocks of circulant matrices, is one of best algebraically constructed LDPC code families with high minimum distance values.

The IP optimizations are performed on a computer with 2.27 GHz Intel Xeon CPU and 12 GB RAM using the CPLEX 12.4 software. Tables 1 and 2 show the minimum distance (d_{min}) and the smallest EPS sizes (the variable nodes numbers in the smallest SS, FAS, AS, and EAS) of LDPC codes of different lengths as well as the time (in seconds) it takes to obtain these results. The optimizations are terminated when the program runs for more than three hours working time or more than 3 GB of memory is utilized. Finding the error-prone structures of a code is an off-line task that does not require real-time processing. However, we still impose a 3 GB memory / three hours time limit, since we have observed that, for codes of practical lengths, if a computation does not terminate within three hours, it tends to fail at terminating in a reasonable time. This happens when the gap between the number of created nodes and the number of solved nodes fails to close. The restriction on the memory the program is allowed to use can also be seen as a restriction on the number of nodes the algorithm can generate. When the number of nodes increases very fast, the program will consume a proportional amount of memory. Estimating when an IP problem will be solved is actually a separate and very important problem in the literature. One popular solution is to utilize the mixed-integer programming (MIP) gap, which is the relative difference between lower and upper bounds on the objective value [23]. We provide a MIP gap analysis of several instances of our problem in Figure 2, where typical solution behavior patterns of our problems are observed. For calculations that are not finished in three hours, the MIP gap tends to stay at high values for a long time and we therefore predict that they won't be finished in a reasonable time.

In Tables 1 and 2, the smallest error-prone substructure sizes are presented as optimal for successfully completed calculations. For incomplete calculations, we also present the lower and upper bounds (LB and UB) on the optimal value based on the state of the solver at the time of termination. As expected, the results show that Tanner codes usually have higher minimum distances than the randomly constructed codes. When it comes to the error prone structures, the same observation is made, i.e., the size of minimum error prone structures is higher in Tanner codes. The size is typically around 18 for stopping sets and 4 for absorbing sets. However, for fully random codes, it is only 2 for both stopping sets and absorbing sets. The PEG algorithm is observed to increase the size of the minimum stopping sets and absorbing sets compared to fully random codes. However, its performance is not enough to catch that of the Tanner codes.

The results given in Table 1 shows that IP is not very efficient in finding minimum distance and minimum SS size. The number of solved nodes for stopping set problems is about half million even for smaller block lengths, which gives us the information about the difficulty of finding minimum SS. Although optimal solutions for the IP problems of SS can be obtained for block lengths up to $1,000$, for larger block lengths, the previously defined termination criteria are met and the solver provides us with bounds on the optimal solution.

On the other hand, the results given in Table 2 demonstrate that the runtime performance is quite high for finding the absorbing set sizes of practical codes. Minimum AS, minimum FAS, and minimum EAS sizes were found very quickly for all of the con-

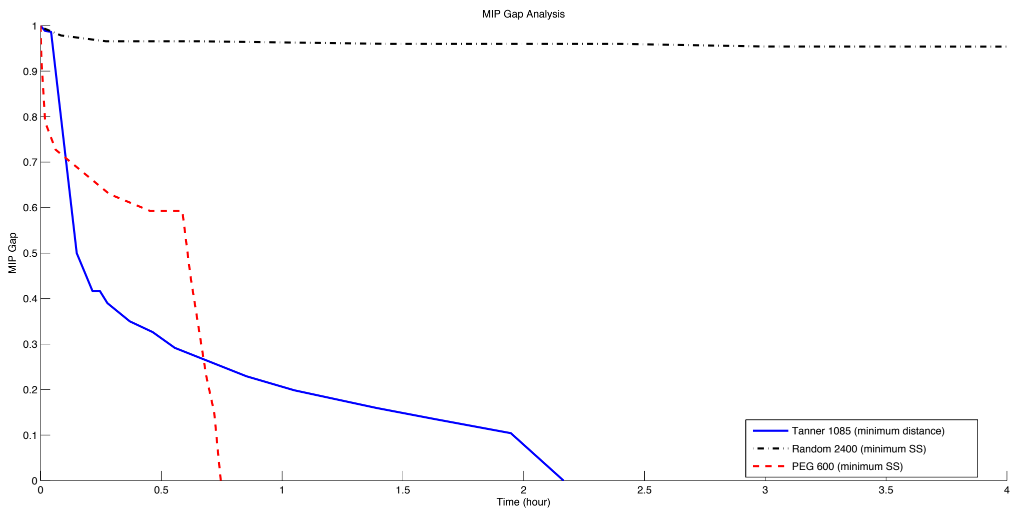


Figure 2: MIP gap analysis of EPS search problems.

sidered codes. It should be noted that, when the block length is less than $1,000$, the computations take mere seconds in most cases. For block lengths of up to $8,000$, optimal solutions are obtained in less than three hours. In fact, only one of the calculations takes a time close to three hours. We should emphasize again that finding and enumerating EPS is an off-line task that is completed in the code design phase and the timing is not as big of a problem as in the real-time decoding implementations (as long as the calculation is successful under given time/memory restrictions). Nevertheless, the proposed IP models are shown to be solved efficiently for finding minimum AS, FAS, and EAS.

6. Conclusions

We presented an efficient, general framework to find and enumerate EPS of any finite-length LDPC code. This includes developing efficient integer programming models to describe and calculate the smallest stopping set size for the BEC, the smallest fully absorbing set, absorbing set, and elementary absorbing set sizes for the BSC and the AWGNC. The obtained results are provably optimal due to the use of integer programming.

The proposed integer programming models require only the knowledge of the parity-check matrix $\mathbf{H}$ and can be efficiently solved for a wide variety of practical code lengths as well as code structures, e.g., regular and irregular LDPC codes, randomly constructed and algebraically constructed codes, etc. With the knowledge of the dominant EPS, the error floor performance of LDPC codes can be estimated, as in the case of [4] and [6]. The obtained result can also be used to improve LDPC code designs, such as done in [24] with maximizing the stopping sets for BEC and in [7] with maximizing trapping sets for BSC. In addition to applications in code design and error performance estimations, the knowledge of the smallest EPS can also be used to generate adaptive cuts for LP-based decoding algorithms. In [25], cycles of the parity-check matrix of an LDPC code are used to improve the LP decoder performance. Since trapping sets are closely related to cycles, the proposed method can be an efficient tool for generating adaptive cutting techniques.

References

[1] R. Gallager, Low-density parity-check codes, IRE Transactions on Information Theory 8 (1) (1962) 21–28.

[2] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, B. Nikolic, Analysis of absorbing sets for array-based LDPC codes, in: IEEE International Conference on Communications, 2007. ICC '07., 2007, pp. 6261–6268.

[3] C. Di, D. Proietti, I. Telatar, T. Richardson, R. Urbanke, Finite-length analysis of low-density parity-check codes on the binary erasure channel, IEEE Transactions on Information Theory 48 (6) (2002) 1570–1579.

[4] T. Richardson, Error floors of LDPC codes, in: Proceedings of the Annual Allerton Conference on Communication Control and Computing, Vol. 41, 2003, pp. 1426–1435.

[5] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, M. Wainwright, Gen03-6: Investigation of error floors of structured low-density parity-check codes by hardware emulation, in: IEEE Global Telecommunications Conference, 2006. GLOBECOM '06., 2006, pp. 1–6.

[6] L. Dolecek, Z. Zhang, M. Wainwright, V. Ananthram, B. Nikolic, Evaluation of the low frame error rate performance of LDPC codes using importance sampling, in: Proc. Information Theory and Applications Workshop, 2007, pp. 202–207.

[7] M. Ivkovic, S. Chilappagari, B. Vasic, Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers, IEEE Transactions on Information Theory 54 (8) (2008) 3763–3768.

[8] A. B. Keha, T. M. Duman, Minumum distance computation of LDPC codes using a branch and cut algorithm, IEEE Transactions on Communications 58 (4) (2010) 1072–1079.

[9] A. McGregor, O. Milenkovic, On the hardness of approximating stopping and trapping sets, IEEE Transactions on Information Theory 56 (4) (2010) 1640–1650.

[10] K. Krishnan, P. Shankar, Computing the stopping distance of a Tanner graph is NP-hard, IEEE Transactions on Information Theory 53 (6) (2007) 2278–2280.

[11] L. Dolecek, P. Lee, Z. Zhang, V. Anantharam, B. Nikolic, M. Wainwright, Predicting error floors of structured ldpc codes: deterministic bounds and estimates, IEEE Journal on Selected Areas in Communications 27 (6) (2009) 908–917.

[12] E. Cavus, B. Daneshrad, A performance improvement and error floor avoidance technique for belief propagation decoding of ldpc codes, in: IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005., Vol. 4, 2005, pp. 2386–2390 Vol. 4.

[13] C. A. Cole, S. G. Wilson, E. K. Hall, T. R. Giallorenzi, A general method for finding low error rates of LDPC codes, CoRR abs/cs/0605051.

[14] M. Karimi, A. Banihashemi, An efficient algorithm for finding dominant trapping sets of LDPC codes, in: Proc. IEEE International Symposium on Turbo Codes and Iterative Information, 2010, pp. 444–448.

[15] G. B. Kyung, C.-C. Wang, Finding the exhaustive list of small fully absorbing sets and designing the corresponding low error-floor decoder, IEEE Transactions on Communications 60 (6) (2012) 1487 –1498.

[16] R. Tanner, A recursive approach to low complexity codes, IEEE Transactions on Information Theory 27 (5) (1981) 533–547.

[17] M. Bazaraa, J. Jarvis, H. Sherali, Linear programming and network flows, 2nd Edition, Wiley, 1990.

[18] F. Hillier, G. Lieberman, Introduction to Operations Research, Introduction to Operations Research, McGraw-Hill Higher Education, 2010.

[19] L. A. Wolsey, Integer Programming, Wiley-Interscience, New York, NY, 1998.

[20] W. Winston, M. Venkataramanan, Introduction to Mathematical Programming: Applications and Algorithms, Brooks/Cole, 2002.

[21] R. Tanner, D. Sridhara, A. Sridharan, T. Fuja, D. Costello Jr, LDPC block and convolutional codes based on circulant matrices, IEEE Transactions on Information Theory 50 (12) (2004) 2966–2984.

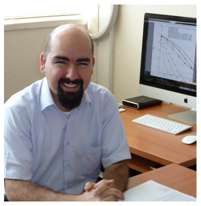
[22] X.-Y. Hu, E. Eleftheriou, D. Arnold, Regular and irregular progressive edge-growth Tanner graphs, IEEE Transactions on Information Theory 51 (1) (2005) 386–398.

[23] O. Y. Özaltin, B. Hunsaker, A. J. Schaefer, Predicting the solution time of branch-and-bound algorithms for mixed-integer programs., INFORMS Journal on Computing 23 (3) (2011) 392–403.

[24] C.-C. Wang, Code annealing and the suppressing effect of the cyclically lifted LDPC code ensemble, in: IEEE Information Theory Workshop, 2006. ITW '06 Chengdu., 2006, pp. 86–90.

[25] M.-H. Taghavi, P. Siegel, Adaptive methods for linear programming decoding, IEEE Transactions on Information Theory 54 (12) (2008) 5396–5410.

Abdullah Sarıduman received the B.Sc. degree in electrical and electronics engineering from TOBB Economy and Technology University, Ankara, Turkey, in 2010. He received the M.Sc. degree in electrical and electronics engineering from Boğaziçi University, Istanbul, in 2013. He is currently working towards a Ph.D. degree at Boğaziçi University, Istanbul, Turkey. His research interests include coding theory, wireless communication, and integer optimization techniques.



Ali E. Pusane received the B.Sc. and M.Sc. degrees in electronics and communications engineering from Istanbul Technical University, Istanbul, Turkey, in 1999 and 2002, respectively, and the M.Sc. degree in electrical engineering, the M.Sc. degree in applied mathematics, and the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 2004, 2006, and 2008, respectively. He was a Visiting Assistant Professor at the Department of Electrical Engineering, University of Notre Dame, during 2008-–2009, after which he joined the Department of Electrical and Electronics Engineering, Boğaziçi University, Istanbul, Turkey, as an Assistant Professor. His research is in coding theory.

Z. Caner Taşkın is an Associate Professor at the Department of Industrial Engineering, Boğaziçi University, Istanbul, Turkey. He received his B.Sc. and M.Sc. degrees in Industrial Engineering from Boğaziçi University in 2003 and 2005, respectively, and his Ph.D. degree in Industrial and Systems Engineering from the University of Florida, Gainesville, FL, in 2009. His main research interests are about integer programming, hybrid decomposition algorithms, and telecommunications network optimization.