Adaptive Cut Generation Algorithm for Improved Linear Programming Decoding of Binary Linear Codes

Xiaojie Zhang, *Student Member, IEEE*, and Paul H. Siegel, *Fellow, IEEE*

***Abstract*—Linear programming (LP) decoding approximates maximum-likelihood (ML) decoding of a linear block code by relaxing the equivalent ML integer programming (IP) problem into a more easily solved LP problem. The LP problem is defined by a set of box constraints together with a set of linear inequalities called "parity inequalities" that are derived from the constraints represented by the rows of a parity-check matrix of the code and can be added iteratively and adaptively. In this paper, we first derive a new necessary condition and a new sufficient condition for a violated parity inequality constraint, or "cut," at a point in the unit hypercube. Then, we propose a new and effective algorithm to generate parity inequalities derived from certain additional redundant parity check (RPC) constraints that can eliminate pseudocodewords produced by the LP decoder, often significantly improving the decoder error-rate performance. The cut-generating algorithm is based upon a specific transformation of an initial parity-check matrix of the linear block code. We also design two variations of the proposed decoder to make it more efficient when it is combined with the new cut-generating algorithm. Simulation results for several low-density parity-check (LDPC) codes demonstrate that the proposed decoding algorithms significantly narrow the performance gap between LP decoding and ML decoding.**

***Index Terms*—Iterative decoding, linear codes, linear programming (LP) decoding, low-density parity-check (LDPC) codes, maximum-likelihood (ML) decoding, pseudocodewords.**

I. Introduction

LOW-DENSITY parity-check (LDPC) codes were first introduced by Gallager in the 1960s [1], together with a class of iterative decoding algorithms. Later, in the 1990s, the rediscovery of LDPC codes by MacKay and Neal [2], [3] launched a period of intensive research on these codes and their decoding algorithms. Significant attention was paid to iterative message-passing (MP) decoders, particularly belief propagation (BP) [4] as embodied by the sum-product algorithm (SPA) [5].

Despite the unparalleled success of iterative decoding in practice, it is quite difficult to analyze the performance of such iterative MP decoders due to the heuristic nature of their message update rules and their local nature. An alternative approach, linear programming (LP) decoding, was introduced by Feldman *et al.* [6] as an approximation to maximum-likelihood (ML) decoding.

Many theoretical and empirical observations suggest similarities between the performance of LP and MP decoding methods. For example, graph-cover decoding can be used as a theoretical tool to show the connection between LP decoding and iterative MP decoding [7].

However, there are some key differences that distinguish LP decoding from iterative MP decoding. One of these differences is that the LP decoder has the *ML certificate property*, i.e., it is detectable if the decoding algorithm fails to find an ML codeword. When it fails to find an ML codeword, the LP decoder finds a noninteger solution, commonly called a *pseudocodeword*. Another difference is that while adding redundant parity checks (RPCs) satisfied by all the codewords can only improve LP decoding, adding RPCs may have a negative effect on MP decoding, especially in the waterfall region, due to the creation of short cycles in the Tanner graph. This property of LP decoding allows improvements by tightening the LP relaxation, i.e., reducing the feasible space of the LP problem by adding more linear constraints from RPCs.

In the original formulation of LP decoding proposed by Feldman *et al.*, the number of constraints in the LP problem is linear in the block length but exponential in the maximum check node degree, and the authors also argued that the number of useful constraints could be reduced to polynomial in code length. The computational complexity of the original LP formulation, therefore, can be prohibitively high, motivating the design of computationally simplified decoding algorithms that can achieve the same error-rate performance with a smaller number of constraints. For example, efficient polynomial-time algorithms can be used for solving the original LP formulation [8]. An alternative LP formulation whose size is linear in the check node degree and code length can also be obtained by changing the graphical representation of the code [9], [10]; namely, all check nodes of high degree are replaced by dendro-subgraphs (trees) with an appropriate number of auxiliary degree-3 check nodes and degree-2 variable nodes. Several

Manuscript received May 03, 2011; revised February 27, 2012; accepted June 08, 2012. Date of publication June 15, 2012; date of current version September 11, 2012. This work was supported in part by the Center for Magnetic Recording Research at the University of California, San Diego, and in part by the National Science Foundation under Grant CCF-0829865. This paper was presented in part at the 2011 IEEE International Symposium on Information Theory.

The authors are with the Department of Electrical and Computer Engineering and the Center for Magnetic Recording Research, University of California, San Diego, La Jolla, CA 92093 USA (e-mail: ericzhang@ucsd.edu; psiegel@ucsd.edu).

Communicated by P. O. Vontobel, Associate Editor for Coding Techniques.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2012.2204955

other low-complexity LP decoders were also introduced in [11], suggesting that LP solvers with complexity similar to the min-sum algorithm and the SPA are feasible.

Another approach is to add linear constraints in an adaptive and selective way during the LP formulation [12]. Such an adaptive linear programming (ALP) decoding approach also allows the adaptive incorporation of linear constraints generated by RPCs into the LP problem, making it possible to reduce the feasible space and improve the system performance. A linear inequality derived from an RPC that eliminates a pseudocodeword solution is referred to as a "cut."

An algorithm proposed in [12] uses a random walk on a subset of the code factor graph to find these RPC cuts. However, the random nature of this algorithm limits its efficiency. In fact, experiments show that the average number of random trials required to find an RPC cut grows exponentially with the length of the code.

Recently, the authors of [13] proposed a separation algorithm (SA) that derives Gomory cuts from the integer programming (IP) formulation of the decoding problem and finds cuts from RPCs which are generated by applying Gaussian elimination to the original parity-check matrix. In [14], a cutting-plane method was proposed to improve the fractional distance of a given binary parity-check matrix—the minimum weight of nonzero vertices of the fundamental polytope—by adding redundant rows obtained by converting the parity-check matrix into row echelon form after a certain column permutation. However, we have observed that the RPCs obtained by the approach in [14] are not able to produce enough cuts to improve the error-rate performance relative to the SA when they are used in conjunction with either ALP decoding or the SA. A detailed survey on mathematical programming approaches for decoding binary linear codes can be found in [15].

In this paper, we greatly improve the error-correcting performance of LP decoding by designing algorithms that can efficiently generate cut-inducing RPCs and find possible cuts from such RPCs. First, we derive a new necessary condition and a new sufficient condition for a parity check to provide a cut at a given pseudocodeword. These conditions naturally suggest an efficient algorithm that can be used to find, for a given pseudocodeword solution to an LP problem, the unique cut (if it exists) among the parity inequalities associated with a parity check. This algorithm was previously introduced by Taghavi *et al.* [16, Algorithm 2] and, independently and in a slightly different form, by Wadayama [17, Fig. 6].

The conditions also serve as the motivation for a new, more efficient adaptive cut-inducing RPC generation algorithm that identifies useful RPCs by performing specific elementary row operations on the original parity-check matrix of the binary linear code. By adding the corresponding linear constraints into the LP problem, we can significantly improve the error-rate performance of the LP decoder, even approaching the ML decoder performance in the high-SNR region for some codes. Finally, we modify the ALP decoder to make it more efficient when being combined with the new cut-generating algorithm. Simulation results demonstrate that the proposed decoding algorithms significantly improve the error-rate performance of the original LP decoder.

The remainder of this paper is organized as follows. In Section II, we review the original formulation of LP decoding and several adaptive LP decoding algorithms. Section III presents the new necessary condition and new sufficient condition for a parity-check to induce a cut, as well as their connection to the efficient cut-search algorithm (CSA). In Section IV, we describe our proposed algorithm for finding RPC-based cuts. Section V presents our simulation results, and Section VI concludes this paper.

II. LP Decoding and Adaptive Variants

A. LP Relaxation of ML Decoding

Consider a binary linear block code $\mathcal{C}$ of length n and a corresponding $m \times n$ parity-check matrix $\mathbf{H}$. A codeword $\mathbf{y} \in \mathcal{C}$ is transmitted across a memoryless binary-input output-symmetric channel, resulting in a received vector $\mathbf{r}$. Assuming that the transmitted codewords are equiprobable, the ML decoder finds the solution to the following optimization problem (see, e.g., [12]):

$$\begin{aligned} &\text{minimize} \quad \gamma^T \mathbf{u} \\ &\text{subject to} \quad \mathbf{u} \in \mathcal{C} \end{aligned} \tag{1}$$

where $u_i \in \{0, 1\}$, and γ is the vector of log-likelihood ratios (LLR) defined as

$$\gamma_i = \log\left(\frac{\Pr(R_i = r_i | u_i = 0)}{\Pr(R_i = r_i | u_i = 1)}\right). \tag{2}$$

Since the ML decoding problem (1) is an IP problem, it is desirable to replace its integrality constraints with a set of linear constraints, transforming the IP problem into a more readily solved LP problem. The desired feasible space of the corresponding LP problem should be the *codeword polytope*, i.e., the convex hull of all the codewords in $\mathcal{C}$. With this, unless the cost vector of the LP decoding problem is orthogonal to a face of the constraint polytope, the optimal solution is one integral vertex of its codeword polytope, in which case it is the same as the output of the ML decoder. When the LP solution is not unique, there is at least one integral vertex corresponding to an ML codeword. However, the number of linear constraints typically needed to represent the codeword polytope increases exponentially with the code length, which makes such a relaxation impractical.

As an approximation to ML decoding, Feldman *et al.* [6], [8] relaxed the codeword polytope to a polytope now known as fundamental polytope [7], denoted as $\mathcal{P}(\mathbf{H})$, which depends on the parity-check matrix $\mathbf{H}$.

Definition 1 (Fundamental Polytope [7]): Let us define

$$\mathcal{C}_j \triangleq \{\mathbf{x} \in \mathbb{F}_2^n | \langle \mathbf{x}, \mathbf{h}_j \rangle = 0 \ (\text{in } \mathbb{F}_2)\} \tag{3}$$

where $\mathbf{h}_j$ is the jth row of the parity-check matrix $\mathbf{H}$ and $1 \leq j \leq m$. Thus, $\mathcal{C}_j$ is the set of all binary vectors satisfying the jth parity-check constraint. We denote by $\text{conv}(\mathcal{C}_j)$ the convex hull of $\mathcal{C}_j$ in $\mathbb{R}^n$, which consists of all possible real convex combinations of the points in $\mathcal{C}_j$, now regarded as points in $\mathbb{R}^n$. The

fundamental polytope $\mathcal{P}(\mathbf{H})$ of the parity-check matrix $\mathbf{H}$ is defined to be the set

$$\mathcal{P}(\mathbf{H}) = \bigcap_{j=1}^{m} \mathrm{conv}(\mathcal{C}_j). \tag{4}$$

Therefore, LP decoding can be written as the following optimization problem:

$$\begin{aligned} &\text{minimize} \quad \gamma^T \mathbf{u} \\ &\text{subject to} \quad \mathbf{u} \in \mathcal{P}(\mathbf{H}). \end{aligned} \tag{5}$$

The solution of the aforementioned LP problem corresponds to a vertex of the fundamental polytope that minimizes the cost function. Since the fundamental polytope has both integral and nonintegral vertices, with the integral vertices corresponding exactly to the codewords of $\mathcal{C}$ [6], [7], if the LP solver outputs an integral solution, it must be a valid codeword and is guaranteed to be an ML solution, which is called the ML certificate property. The nonintegral solutions are called pseudocodewords. Since the fundamental polytope is a function of the parity-check matrix $\mathbf{H}$ used to represent the code $\mathcal{C}$, different parity-check matrices for $\mathcal{C}$ may have different fundamental polytopes. Therefore, a given code has many possible LP-based relaxations, and some may be better than others when used for LP decoding.

The fundamental polytope can also be described by a set of linear inequalities, obtained as follows [6]. First of all, for a point $\mathbf{u}$ within the fundamental polytope, it should satisfy the box constraints such that $0 \le u_i \le 1$, for $i = 1, \ldots, n$. Then, let $\mathcal{N}(j) \subseteq \{1, 2, \ldots, n\}$ be the set of neighboring variable nodes of the check node j in the Tanner graph, that is, $\mathcal{N}(j) = \{i : H_{j,i} = 1\}$ where $H_{j,i}$ is the element in the jth row and ith column of the parity-check matrix, $\mathbf{H}$. For each row $j = 1, \ldots, m$ of the parity-check matrix, corresponding to a check node in the associated Tanner graph, the linear inequalities used to form the fundamental polytope $\mathcal{P}(\mathbf{H})$ are given by

$$\sum_{i \in \mathcal{V}} (1 - u_i) + \sum_{i \in \mathcal{N}(j) \setminus \mathcal{V}} u_i \ge 1, \ \forall\, \mathcal{V} \subseteq \mathcal{N}(j), \ \text{with} \ |\mathcal{V}| \ \text{odd} \tag{6}$$

where for a set $\mathcal{X}$, $|\mathcal{X}|$ denotes its cardinality. It is easy to see that (6) is equivalent to

$$\sum_{i \in \mathcal{V}} u_i - \sum_{i \in \mathcal{N}(j) \setminus \mathcal{V}} u_i \le |\mathcal{V}| - 1, \ \forall\, \mathcal{V} \subseteq \mathcal{N}(j), \ \text{with} \ |\mathcal{V}| \ \text{odd}. \tag{7}$$

Note that, for each check node j, the corresponding inequalities in (6) or (7) and the linear box constraints exactly describe the convex hull of the set $\mathcal{C}_j$.

The linear constraints in (6) [and therefore also (7)] are referred to as *parity inequalities*, which are also known as *forbidden set inequalities* [6]. It can be easily verified that these linear constraints are equivalent to the original parity-check constraints when each u_i takes on binary values only.

Proposition 1 ([6, Th. 4]): The parity inequalities of the form (6) derived from all rows of the parity-check matrix $\mathbf{H}$ and the box constraints completely describe the fundamental polytope $\mathcal{P}(\mathbf{H})$.

With this, LP decoding can also be formulated as follows:

$$\begin{aligned} &\text{minimize} \quad \gamma^T \mathbf{u} \\ &\text{subject to} \quad 0 \le u_i \le 1, \ \text{for all} \ \ i; \\ &\qquad \sum_{i \in \mathcal{V}} (1 - u_i) + \sum_{i \in \mathcal{N}(j) \setminus \mathcal{V}} u_i \ge 1 \\ &\qquad \text{for all} \ \ j, \mathcal{V} \subseteq \mathcal{N}(j), \ \text{with} \ \ |\mathcal{V}| \ \text{odd}. \end{aligned} \tag{8}$$

In the following parts of this paper, we refer to the aforementioned formulation of LP decoding problem based on the fundamental polytope of the original parity-check matrix as the *original* LP decoding.

B. ALP Decoding

In the original formulation of LP decoding presented in [6], every check node j generates $2^{|\mathcal{N}(j)|-1}$ parity inequalities that are used as linear constraints in the LP problem described in (8). The total number of constraints and the complexity of the original LP decoding problem grow exponentially with the maximum check node degree. So, even for binary linear codes with moderate check degrees, the number of constraints in the original LP decoding could be prohibitively large. In the literature, several approaches to reducing the complexity of the original LP formulation have been described [8]–[12]. We will use ALP decoding [12] as the foundation of the improved LP decoding algorithms presented in later sections. The ALP decoder exploits the structure of the LP decoding problem, reflected in the statement of the following lemma.

Lemma 1 ([12, Th. 1]): If at any given point $\mathbf{u} \in [0,1]^n$ one of the parity inequalities introduced by a check node j is violated, the rest of the parity inequalities from this check node are satisfied with strict inequality.

Definition 2: Given a parity-check node j, a set $\mathcal{V} \subseteq \mathcal{N}(j)$ of odd cardinality, and a vector $\mathbf{u} \in [0,1]^n$ such that the corresponding parity inequality of the form (6) or (7) does not hold, we say that the constraint is *violated* or, more succinctly, a *cut* at $\mathbf{u}$.[1]

In [12], an efficient algorithm for finding cuts at a vector $\mathbf{u} \in [0,1]^n$ was presented. It relies on the observation that violation of a parity inequality (7) at $\mathbf{u}$ implies that

$$|\mathcal{V}| - 1 < \sum_{i \in \mathcal{V}} u_i \le |\mathcal{V}| \tag{9}$$

and

$$0 \le \sum_{i \in \mathcal{N}(j) \setminus \mathcal{V}} u_i < u_v, \ \text{for all} \ \ v \in \mathcal{V} \tag{10}$$

where $\mathcal{V}$ is an odd-sized subset of $\mathcal{N}(j)$.

Given a parity check j, the algorithm first puts its neighboring variables in $\mathbf{u}$ into nonincreasing order, i.e., $u_{j_1} \ge \ldots \ge u_{j_n}$, for $u_{j_i} \in \mathcal{N}(j)$. It then successively considers subsets of odd cardinality having the form $\mathcal{V} = \{u_{j_1}, \ldots, u_{j_{2k+1}}\} \subseteq \mathcal{N}(j)$, increasing the size of $\mathcal{V}$ by 2 each step, until a cut (if one exists) is

[1] In the terminology of [15], if (7) does not hold for a pseudocodeword $\mathbf{u}$, then the vector $(\mathbf{r}, t) \in \mathbb{R}^n \times \mathbb{R}$, where $r_i = 1$ for all $i \in \mathcal{V}$, $r_i = -1$ for all $i \in \mathcal{N}(j) \setminus \mathcal{V}$, $r_i = 0$ otherwise, and $t = |\mathcal{V}| - 1$, is a *valid cut*, separating $\mathbf{u}$ from the codeword polytope.

found. This algorithm can find a cut among the constraints corresponding to a check node j by examining at most $|\mathcal{N}(j)|/2$ inequalities, rather than exhaustively checking all $2^{|\mathcal{N}(j)|-1}$ inequalities in the original formulation of LP decoding.

The ALP decoding algorithm starts by solving the LP problem with the same objective function as (1), but with only the following constraints:

$$\begin{cases} 0 \le u_i, & \text{if } \gamma_i \ge 0 \\ u_i \le 1, & \text{if } \gamma_i < 0. \end{cases} \tag{11}$$

The solution of this initial LP problem can be obtained simply by making a hard decision on the components of a received vector. The ALP decoding algorithm starts with this point, searches every check node for cuts, adds all the cuts found during the search as constraints into the LP problem, and solves it again. This procedure is repeated until an optimal integer solution is generated or no more cuts can be found (see [12] for more details). Adaptive LP decoding has exactly the same error-correcting performance as the original LP decoding.

III. Cut Conditions

In this section, we derive a necessary condition and a sufficient condition for a parity inequality to be a cut at $\mathbf{u} \in [0,1]^n$. We also show their connection to the efficient cut-search algorithm (CSA) proposed by Taghavi *et al.* [16, Algorithm 2] and Wadayama [17, Fig. 6]. This algorithm is more efficient than the search technique from [12] that was mentioned in Section II.

Consider the original parity inequalities in (6) given by Feldman *et al.* in [6]. If a parity inequality derived from check node j induces a cut at $\mathbf{u}$, the cut can be written as

$$\sum_{i \in \mathcal{V}} (1 - u_i) + \sum_{i \in \mathcal{N}(j) \setminus \mathcal{V}} u_i < 1, \tag{12}$$

for some $\mathcal{V} \subseteq \mathcal{N}(j)$ with $|\mathcal{V}|$ odd.

From (12) and Lemma 1, we can derive the following necessary condition for a parity-check constraint to induce a cut.

Theorem 1: Given a nonintegral vector $\mathbf{u}$ and a parity check j, let $\mathcal{S} = \{i \in \mathcal{N}(j) | 0 < u_i < 1\}$ be the set of nonintegral neighbors of j in the Tanner graph, and let $\mathcal{T} = \{i \in \mathcal{N}(j) | u_i > \frac{1}{2}\}$. A necessary condition for parity check j to induce a cut at $\mathbf{u}$ is

$$\sum_{i \in \mathcal{T}} (1 - u_i) + \sum_{i \in \mathcal{N}(j) \setminus \mathcal{T}} u_i < 1. \tag{13}$$

This is equivalent to

$$\sum_{i \in \mathcal{S}} \left| \frac{1}{2} - u_i \right| > \frac{1}{2} \cdot |\mathcal{S}| - 1 \tag{14}$$

where, for $x \in \mathbb{R}$, $|x|$ denotes the absolute value.

Proof: For a given vector $\mathbf{u}$ and a subset $\mathcal{X} \subseteq \mathcal{N}(j)$, define the function

$$g(\mathcal{X}) = \sum_{i \in \mathcal{X}} (1 - u_i) + \sum_{i \in \mathcal{N}(j) \setminus \mathcal{X}} u_i.$$

If parity check j incudes a cut at $\mathbf{u}$, there must be a set $\mathcal{V} \subseteq \mathcal{N}(j)$ of odd cardinality such that (12) holds. This means that $g(\mathcal{V}_{\text{cut}}) < 1$. Now, it is easy to see that the set $\mathcal{T}$ minimizes the function $g(\mathcal{X})$, from which it follows that $g(\mathcal{T}) \le g(\mathcal{V}_{\text{cut}}) < 1$. Therefore, inequality (13) must hold in order for parity check j to induce a cut.

For $\frac{1}{2} \le u_i \le 1$, we have

$$\frac{1}{2} - \left| \frac{1}{2} - u_i \right| = \frac{1}{2} - \left(u_i - \frac{1}{2} \right) = 1 - u_i,$$

and for $0 \le u_i \le \frac{1}{2}$, we have

$$\frac{1}{2} - \left| \frac{1}{2} - u_i \right| = \frac{1}{2} - \left(\frac{1}{2} - u_i \right) = u_i.$$

Hence, (13) can be rewritten as

$$\sum_{i \in \mathcal{S}} \left(\frac{1}{2} - \left| \frac{1}{2} - u_i \right| \right) < 1$$

or equivalently

$$\frac{1}{2} \cdot |\mathcal{S}| - \sum_{i \in \mathcal{S}} \left| \frac{1}{2} - u_i \right| < 1$$

which implies inequality (14). ■

Remark 1: Theorem 1 shows that to see whether a parity-check node could provide a cut at a pseudocodeword $\mathbf{u}$ we only need to examine its fractional neighbors.

Reasoning similar to that used in the proof of Theorem 1 yields a sufficient condition for a parity-check node to induce a cut at $\mathbf{u}$.

Theorem 2: Given a nonintegral vector $\mathbf{u}$ and a parity check j, let $\mathcal{S} = \{i \in \mathcal{N}(j) | 0 < u_i < 1\}$ and $\mathcal{T} = \{i \in \mathcal{N}(j) | u_i > \frac{1}{2}\}$. If the inequality

$$\sum_{i \in \mathcal{T}} (1 - u_i) + \sum_{i \in \mathcal{N}(j) \setminus \mathcal{T}} u_i + 2 \cdot \min_{i \in \mathcal{S}} \left| \frac{1}{2} - u_i \right| < 1 \tag{15}$$

holds, there must be a violated parity inequality derived from parity check j. This sufficient condition can be written as

$$\sum_{i \in \mathcal{S}} \left| \frac{1}{2} - u_i \right| - 2 \cdot \min_{i \in \mathcal{S}} \left| \frac{1}{2} - u_i \right| > \frac{1}{2} \cdot |\mathcal{S}| - 1. \tag{16}$$

Proof: Lemma 1 implies that if parity check j gives a cut at $\mathbf{u}$, then there is at most one odd-sized set $\mathcal{V} \subseteq \mathcal{N}(j)$ that satisfies (12). From the proof of Theorem 1, we have $g(\mathcal{T}) \le g(\mathcal{X})$ for all $\mathcal{X} \subseteq \mathcal{N}(j)$. If $|\mathcal{T}|$ is even, we need to find one element $i^* \in \mathcal{N}(j)$ such that inserting it into or removing it from $\mathcal{T}$ would result in the minimum increment to the value of

$g(\mathcal{T})$. Obviously, $i^* = \arg\min_{i \in \mathcal{N}(j)} \left|\frac{1}{2} - u_i\right|$, and the increment is $2 \cdot \left|\frac{1}{2} - u_{i^*}\right|$. If more than one i minimizes the expression $\left|\frac{1}{2} - u_i\right|$, we choose one arbitrarily as i^*. Hence, setting

$$\mathcal{V} = \begin{cases} \mathcal{T} \backslash \{i^*\}, & \text{if } i^* \in \mathcal{T} \\ \mathcal{T} \cup \{i^*\}, & \text{if } i^* \notin \mathcal{T} \end{cases}$$

we have $g(\mathcal{V}) = g(\mathcal{T}) + 2 \cdot \left|\frac{1}{2} - u_{i^*}\right| \geq g(\mathcal{T})$. If inequality (15) holds, then $g(\mathcal{T}) \leq g(\mathcal{V}) < 1$. Since either $|\mathcal{T}|$ or $|\mathcal{V}|$ is odd, (15) is a sufficient condition for parity-check constraint j to induce a cut at $\mathbf{u}$. Arguing as in the latter part of the proof of Theorem 1, it can be shown that (15) is equivalent to (16). ■

Theorems 1 and 2 provide a necessary condition and a sufficient condition, respectively, for a parity-check node to produce a cut at any given vector $\mathbf{u}$. It is worth pointing out that (13) becomes a necessary and sufficient condition for a parity check to produce a cut when $|\mathcal{T}|$ is odd, and (15) becomes a necessary and sufficient condition when $|\mathcal{T}|$ is even. Together, they suggest a highly efficient technique for finding cuts, the CSA described in Algorithm 1. If there is a violated parity inequality, the CSA returns the set $\mathcal{V}$ corresponding to the cut; otherwise, it returns an empty set.

Algorithm 1 Cut-Search Algorithm (CSA)

Input: parity-check node j and vector $\mathbf{u}$
Output: variable node set $\mathcal{V}$
1: $\mathcal{V} \leftarrow \mathcal{T} = \{i \in \mathcal{N}(j) | u_i > \frac{1}{2}\}$ and $\mathcal{S} \leftarrow \{i \in \mathcal{N}(j) | 0 < u_i < 1\}$
2: **if** $|\mathcal{V}|$ is even **then**
3: **if** $\mathcal{S} \neq \emptyset$ **then**
4: $i^* \leftarrow \arg\min_{i \in \mathcal{S}} \left|\frac{1}{2} - u_i\right|$
5: **else**
6: $i^* \leftarrow$ arbitrary $i \in \mathcal{N}(j)$
7: **end if**
8: **if** $i^* \in \mathcal{V}$ **then**
9: $\mathcal{V} \leftarrow \mathcal{V} \setminus \{i^*\}$
10: **else**
11: $\mathcal{V} \leftarrow \mathcal{V} \cup \{i^*\}$
12: **end if**
13: **end if**
14: **if** $\sum_{i \in \mathcal{V}} (1 - u_i) + \sum_{i \in \mathcal{N}(j) \backslash \mathcal{V}} u_i < 1$ **then**
15: Found the violated parity inequality on parity-check node j
16: **else**
17: There is no violated parity inequality on parity-check node j
18: $\mathcal{V} \leftarrow \emptyset$
19: **end if**
20: **return** $\mathcal{V}$

As mentioned previously, the CSA was used by Taghavi *et al.* [16, Algorithm 2] in conjunction with ALP decoding, and by Wadayama [17, Fig. 6] as a feasibility check in the context of interior point decoding. In addition to providing another perspective on the CSA, the necessary condition and sufficient condition proved in Theorems 1 and 2, respectively, serve as the basis for a new adaptive approach to finding cut-inducing RPCs, as described in Section IV.

IV. LP Decoding With Adaptive Cut-Generating Algorithm

A. Generating RPCs

Although the addition of a redundant row to a parity-check matrix does not affect the $\mathbb{F}_2$-nullspace and, therefore, the linear code it defines, different parity-check matrix representations of a linear code may give different fundamental polytopes underlying the corresponding LP relaxation of the ML decoding problem. This fact inspires the use of cutting-plane techniques to improve the error-correcting performance of the original LP and ALP decoders. Specifically, when the LP decoder gives a nonintegral solution (i.e., a pseudocodeword), we try to find the RPCs that introduce cuts at that point, if such RPCs exist. The cuts obtained in this manner are called *RPC cuts*. The effectiveness of this method depends on how closely the new relaxation approximates the ML decoding problem, as well as on the efficiency of the technique used to search for the cut-inducing RPCs.

An RPC can be obtained by modulo-2 addition of some of the rows of the original parity-check matrix, and this new check introduces a number of linear constraints that may give a cut. In [12], a random walk on a cycle within the subgraph defined by the nonintegral entries in a pseudocodeword served as the basis for a search for RPC cuts. However, there is no guarantee that this method will find a cut (if one exists) within a finite number of iterations. In fact, the average number of random trials needed to find an RPC cut grows exponentially with the code length.

The IP-based SA in [13] performs Gaussian elimination on a submatrix comprising the columns of the original parity-check matrix that correspond to the nonintegral entries in a pseudocodeword in order to get RPCs. In [14], the RPCs that potentially provide cutting planes are obtained by transforming a column-permuted version of the submatrix into row echelon form. The chosen permutation organizes the columns according to descending order of their associated nonintegral pseudocodeword entries, with the exception of the column corresponding to the largest nonintegral entry, which is placed in the rightmost position of the submatrix [14, p. 1010]. This approach was motivated by the fact that a parity check j provides a cut at a pseudocodeword if there exists a variable node in $\mathcal{N}(j)$ whose value is greater than the sum of the values of all of the other neighboring variable nodes [14, Lemma 2]. However, when combined with ALP decoding, the resulting "cutting-plane algorithm" does not provide sufficiently many cuts to surpass the SA in error-rate performance.

Motivated by the new derivation of the CSA based on the conditions in Theorems 1 and 2, we next propose a new algorithm for generating cut-inducing RPCs. When used with ALP decoding, the cuts have been found empirically to achieve near-ML decoding performance in the high-SNR region for several short-to-moderate length LDPC codes. However, application of these new techniques to codes with larger block lengths proved to be prohibitive computationally, indicating that further work is required to develop practical methods for enhanced LP decoding of longer codes.

Given a nonintegral solution of the LP problem, we can see from Theorems 1 and 2 that an RPC with a small number of nonintegral neighboring variable nodes may be more likely to satisfy the necessary condition for providing a cut at the pseudocodeword. Moreover, the nonintegral neighbors should have values either close to 0 or close to 1; in other words, they should be as far from $\frac{1}{2}$ as possible.

Let $\mathbf{p} = (p_1, p_2, \ldots, p_n) \in [0,1]^n$ be a pseudocodeword solution to LP decoding, with a nonintegral positions, b 0s, and $n-a-b$ 1s. We first group entries of $\mathbf{p}$ according to whether their values are nonintegral, 0, or 1. Then, we sort the nonintegral positions in ascending order according to the value of $\left|\frac{1}{2} - p_i\right|$ and define the permuted vector $\mathbf{p}' = \Pi(\mathbf{p})$ satisfying the following ordering:

$$\left|\frac{1}{2} - p'_1\right| \leq \cdots \leq \left|\frac{1}{2} - p'_a\right|, \tag{17}$$

and

$$p'_{a+1} = \cdots = p'_{a+b} = 0,$$
$$p'_{a+b+1} = \cdots = p'_n = 1.$$

By applying the same permutation Π to the columns of the original parity-check matrix $\mathbf{H}$, we get

$$\mathbf{H}' \stackrel{\triangle}{=} \Pi(\mathbf{H}) = \left(\mathbf{H}^{(\mathrm{f})} | \mathbf{H}^{(0)} | \mathbf{H}^{(1)}\right) \tag{18}$$

where $\mathbf{H}^{(\mathrm{f})}$, $\mathbf{H}^{(0)}$, and $\mathbf{H}^{(1)}$ consist of columns of $\mathbf{H}$ corresponding to positions of $\mathbf{p}'$ with nonintegral values, 0s, and 1s, respectively.

The following familiar definition from matrix theory will be useful [18, p. 10].

Definition 3: A matrix is in *reduced row echelon* form if its nonzero rows (i.e., rows with at least one nonzero element) are above any all-zero rows, and the leading entry (i.e., the first nonzero entry from the left) of a nonzero row is the only nonzero entry in its column and is always strictly to the right of the leading entry of the row above it.

By applying a suitable sequence of elementary row operations Φ (over $\mathbb{F}_2$) to $\mathbf{H}'$, we get

$$\bar{\mathbf{H}} \stackrel{\triangle}{=} \Phi(\mathbf{H}') = \left(\bar{\mathbf{H}}^{(\mathrm{f})} | \bar{\mathbf{H}}^{(0)} | \bar{\mathbf{H}}^{(1)}\right) \tag{19}$$

where $\bar{\mathbf{H}}^{(\mathrm{f})}$ is in reduced row echelon form. Applying the inverse permutation Π^{-1} to columns of $\bar{\mathbf{H}}$, we get an equivalent parity-check matrix

$$\tilde{\mathbf{H}} = \Pi^{-1}(\bar{\mathbf{H}}) \tag{20}$$

whose rows are likely to be cut-inducing RPCs, for the reasons stated previously.

Multiple nonintegral positions in the pseudocodeword $\mathbf{p}$ could have values with the same distance from $\frac{1}{2}$, i.e., $|\frac{1}{2} - p_i| = |\frac{1}{2} - p_j|$ for some $i \neq j$. In such a case, the ordering of the nonintegral positions in (17) is not uniquely determined. Hence, the set of RPCs generated by operations (18) – (20) may depend upon the particular ordering reflected in the permutation Π. Nevertheless, if the decoder uses a fixed, deterministic sorting rule such as, for example, a stable sorting algorithm, then the decoding error probability will be independent of the transmitted codeword.

The next theorem describes a situation in which a row of $\tilde{\mathbf{H}}$ is guaranteed to provide a cut.

Theorem 3: If there exists a weight-one row in submatrix $\bar{\mathbf{H}}^{(\mathrm{f})}$, the corresponding row of the equivalent parity-check matrix $\tilde{\mathbf{H}}$ is a cut-inducing RPC.

Proof: Given a pseudocodeword $\mathbf{p}$, suppose the jth row of submatrix $\bar{\mathbf{H}}^{(\mathrm{f})}$ has weight 1 and the corresponding nonintegral position in $\mathbf{p}$ is p_i. Since it is the only nonintegral position in $\mathcal{N}(j)$, the left-hand side of (16) is equal to $-\left|\frac{1}{2} - p_i\right|$. Since $0 < p_i < 1$, this is larger than $-\frac{1}{2}$, the right-hand side. Hence, according to Theorem 2, RPC j satisfies the sufficient condition for providing a cut. In other words, there must be a violated parity inequality induced by RPC j. ■

Remark 2: Theorem 3 is equivalent to [13, Theorem 3.3]. The proof of the result shown here, though, is considerably simpler, thanks to the application of Theorem 2.

Although Theorem 3 only ensures a cut for rows with weight 1 in submatrix $\bar{\mathbf{H}}^{(\mathrm{f})}$, rows in $\bar{\mathbf{H}}^{(\mathrm{f})}$ of weight greater than 1 may also provide RPC cuts. Hence, the CSA should be applied on every row of the redundant parity-check matrix $\tilde{\mathbf{H}}$ to search for all possible RPC cuts. The approach of generating a redundant parity-check matrix $\tilde{\mathbf{H}}$ based on a given pseudocodeword and applying the CSA on each row of this matrix is called adaptive cut generation (ACG). Combining ACG with ALP decoding, we obtain the ACG-ALP decoding algorithm described in Algorithm 2. Beginning with the original parity-check matrix, the algorithm iteratively applies ALP decoding. When a point is reached when no further cuts can be produced from the original parity-check matrix, the ACG technique is invoked to see whether any RPC cuts can be generated. The ACG-ALP decoding iteration stops when no more cuts can be found either from the original parity-check matrix or in the form of RPCs.

Algorithm 2 ALP with adaptive cut-generation (ACG-ALP) decoding algorithm

Input: cost vector γ, original parity-check matrix $\mathbf{H}$
Output: Optimal solution of current LP problem
1: Initialize the LP problem with the constraints in **(11)**.
2: Solve the current LP problem, and get optimal solution x^*.
3: Apply **Algorithm 1 (CSA)** on each row of $\mathbf{H}$.
4: **if** No cut is found **and** x^* is nonintegral **then**
5: Construct $\tilde{\mathbf{H}}$ associated with x^* according to **(18)–(20)**.
6: Apply **Algorithm 1 (CSA)** to each row of $\tilde{\mathbf{H}}$.
7: **end if**
8: **if** No cut is found **then**
9: Terminate.
10: **else**
11: Add cuts that are found into the LP problem as constraints, and go to line 2.
12: **end if**

B. Reducing the Number of Constraints in the LP Problem

In the ALP decoding, the number of constraints in the LP problem grows as the number of iterations grows, increasing the complexity of solving the LP problem. For ACG-ALP decoding, this problem becomes more severe since the algorithm generates additional RPC cuts and uses more iterations to successfully decode inputs on which the ALP decoder has failed.

From Lemma 1, we know that a binary parity-check constraint can provide at most one cut. Hence, if a binary parity check gives a cut, all other linear inequalities introduced by this parity check in previous iterations can be removed from the LP problem. The implementation of this observation leads to a modified ALP (MALP) decoder referred to as the MALP-A decoder [16]. This decoder improves the efficiency of ALP decoding, where only cuts associated with the original parity-check matrix are used. However, with ACG-ALP decoding, different RPCs may be generated adaptively in every iteration and most of them give only one cut throughout the sequence of decoding iterations. As a result, when MALP-A decoding is combined with the ACG technique, only a small number of constraints are removed from the LP problem, and the decoding complexity is only slightly reduced.

Definition 4: A linear inequality constraint of the form $\mathbf{a}^T\mathbf{x} \geq b$ is called *active* at point $\mathbf{x}^*$ if it holds with equality, i.e., $\mathbf{a}^T\mathbf{x}^* = b$, and is called *inactive* otherwise.

For an LP problem with a set of linear inequality constraints, the optimal solution $\mathbf{x}^* \in [0,1]^n$ is a vertex of the polytope formed by the hyperplanes corresponding to all active constraints. In other words, if we set up an LP problem with only those active constraints, the optimal solution remains the same. Therefore, a simple and intuitive way to reduce the number of constraints is to remove all inactive constraints from the LP problem at the end of each iteration, regardless of whether or not the corresponding binary parity check generates a cut. This approach is called MALP-B decoding [16]. By combining the ACG technique and the MALP-B algorithm, we obtain the ACG-MALP-B decoding algorithm. It is similar to the ACG-ALP algorithm described in Algorithm 2 but includes one additional step that removes all inactive constraints from the LP problem, as indicated in Line 3 of Algorithm 3.

Algorithm 3 ACG-MALP-B/C Decoding Algorithm

Input: cost vector γ, original parity-check matrix $\mathbf{H}$
Output: Optimal solution of current LP problem
1: Initialize LP problem with the constraints in **(11)**.
2: Solve the current LP problem, get optimal solution x^*.
3: ACG-MALP-B only: remove all inactive constraints from the LP problem.
4: ACG-MALP-C only: remove inactive constraints that have above-average slack values from the LP problem.
5: Apply **CSA** only on rows of $\mathbf{H}$ that have not introduced constraints.
6: **if** No cut is found **and** x^* is nonintegral **then**
7: Construct $\tilde{\mathbf{H}}$ according to x^*.
8: Apply **CSA** on each row of $\tilde{\mathbf{H}}$.
9: **end if**
10: **if** No cut is found **then**
11: Terminate.
12: **else**
13: Add found cuts into LP problem as constraints, and go to line 2.
14: **end if**

Since adding further constraints into an LP problem reduces the feasible space, the minimum value of the cost function is nondecreasing as a function of the number of iterations. In our computer simulations, the ACG-MALP-B decoding algorithm was terminated when no further cuts could be found. (See Fig. 6 for statistics on the average number of iterations required to decode one codeword of the (155,64) Tanner LDPC code.)

In our implementation of both MALP-B and ACG-MALP-B decoding, we have noticed that a considerable number of the constraints deleted in previous iterations are added back into the LP problem in later iterations, and, in fact, many of them are added and deleted several times. We have observed that MALP-B-based decoding generally takes more iterations to decode a codeword than ALP-based decoding, resulting in a tradeoff between the number of iterations and the size of the constituent LP problems. MALP-B-based decoding has the largest number of iterations and the smallest LP problems to solve in each iteration, while ALP-based decoding has a smaller number of iterations but larger LP problems.

Although it is difficult to know in advance which inactive constraints might become cuts in later iterations, there are several ways to find a better tradeoff between the MALP-B and ALP techniques to speed up LP decoding. This tradeoff, however, is highly dependent on the LP solver used in the implementation. For example, we used the simplex solver from the open-source GNU linear programming kit (GLPK) [19], and found that the efficiency of iterative ALP-based decoders is closely related to the total number of constraints used to decode one codeword, i.e., the sum of the number of constraints used in all iterations. This suggests a new criterion for the removal of inactive constraints whose implementation we call the MALP-C decoder.

In MALP-C decoding, instead of removing all inactive constraints from the LP problem in each iteration, we remove only the linear inequality constraints with slack variables that have above-average values, as indicated in Line 4 of Algorithm 3. The ACG-MALP-B and ACG-MALP-C decoding algorithms are both described in Algorithm 3, differing only in the use of Line 3 or Line 4. Although all three of the adaptive variations of LP decoding discussed in this paper—ALP, MALP-B, and MALP-C—have the exact same error-rate performance as the original LP decoder, they may lead to different decoding results for a given received vector when combined with the ACG technique, as shown in Section V.

V. Numerical Results

To demonstrate the improvement offered by our proposed decoding algorithms, we compared their error-correcting performance to that of ALP decoding (which, again, has the same performance as the original LP decoding), BP decoding (two cases, using the SPA with a maximum of 100 iterations and 1000 iterations, respectively), the SA [13], the random-walk-based RPC search algorithm [12], and ML decoding for various LDPC codes on the additive white Gaussian noise (AWGN) channel. We use the simplex algorithm from the open-source GLPK [19] as our LP solver. The LDPC codes we evaluated are MacKay's rate-$\frac{1}{2}$, (3,6)-regular LDPC codes with lengths 96 and 408, respectively [20]; a rate-$\frac{1}{4}$, (3,4)-regular LDPC code of length

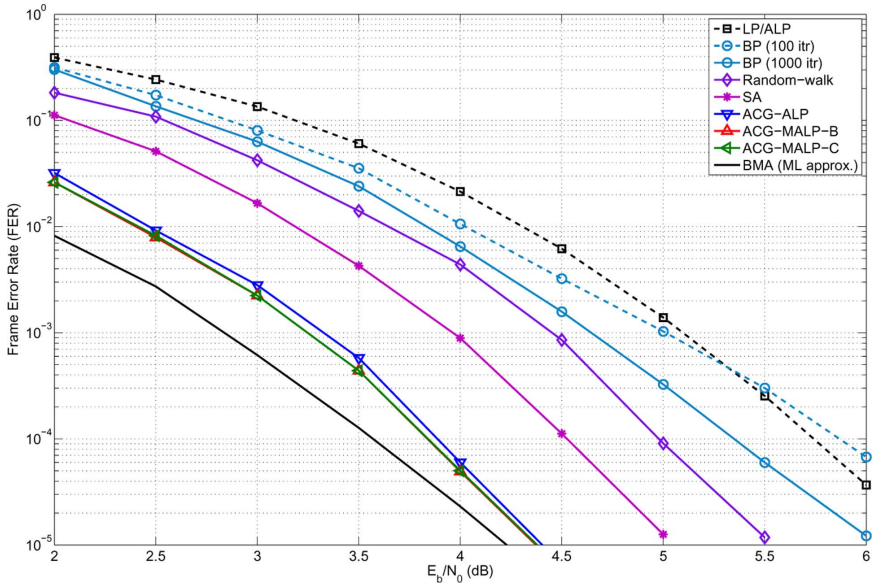

Fig. 1. FER versus E_b/N_0 for random (3,4)-regular LDPC code of length 100 on the AWGN channel.

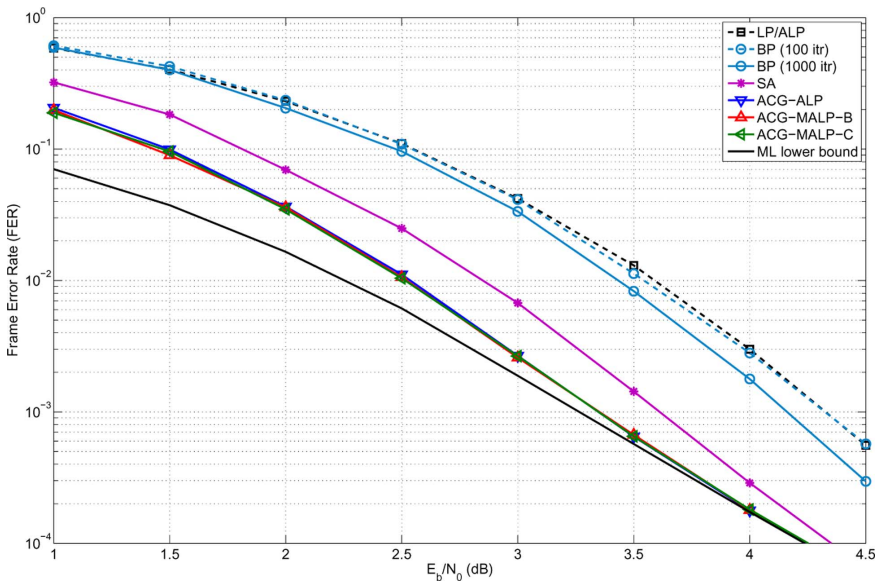

Fig. 2. FER versus E_b/N_0 for MacKay's random (3,6)-regular LDPC code of length 96 on the AWGN channel.

100; the rate-$\frac{2}{5}$, (3,5)-regular Tanner code of length 155 [21]; and a rate-0.89, (3,27)-regular high-rate LDPC code of length 999 [20].

The proposed ACG-ALP, ACG-MALP-B, and ACG-MALP-C decoding algorithms are all based on the underlying cut-searching algorithm (Algorithm 1) and the adaptive cut-generation technique of Section IV-A. Therefore, their error-rate performance is very similar. However, their performance may not be identical, because cuts are found adaptively from the output pseudocodewords in each iteration and the different sets of constraints used in the three proposed algorithms may lead to different solutions of the corresponding LP problems.

In our simulation, the LP solver uses double-precision floating-point arithmetic, and therefore, due to this limited numerical resolution, it may round some small nonzero vector coordinate values to 0 or output small nonzero values for vector coordinates which should be 0. Similar rounding errors may occur for coordinate values close to 1. Coordinates whose values get rounded to integers by the LP solver might lead to some "false" cuts—parity inequalities not actually violated by the exact LP solution. This is because such rounding by the LP solver would decrease the left-hand side of parity inequality (6). On the other hand, when coordinates that should have integer values are given nonintegral values, the resulting errors would increase the left-hand side of parity inequality (6), causing some cuts to be missed. Moreover, this would also increase the size of the submatrix $\mathbf{H}^{(\mathrm{f})}$ in (18), leading to higher complexity for the ACG-ALP decoding algorithm.

To avoid such numerical problems in our implementation of the CSA, we used $1 - 10^{-6}$ instead of 1 on the right-hand side of the inequality in Line 14 of Algorithm 1. Whenever the LP

TABLE I
FRAME ERRORS OF ACG-ALP DECODER ON MACKAY'S RANDOM (3,6)-REGULAR LDPC CODE OF LENGTH 96 ON THE AWGN CHANNEL

E_b/N_0 (dB)	Transmitted Frames	Error Frames	Pseudocodewords	Incorrect Codewords
3.0	1,136,597	3,000	857	2,143
3.5	4,569,667	3,000	395	2,605
4.0	16,724,921	3,000	103	2,897
4.5	54,952,664	3,000	12	2,988
5.0	185,366,246	3,000	0	3,000
5.5	665,851,530	3,000	0	3,000

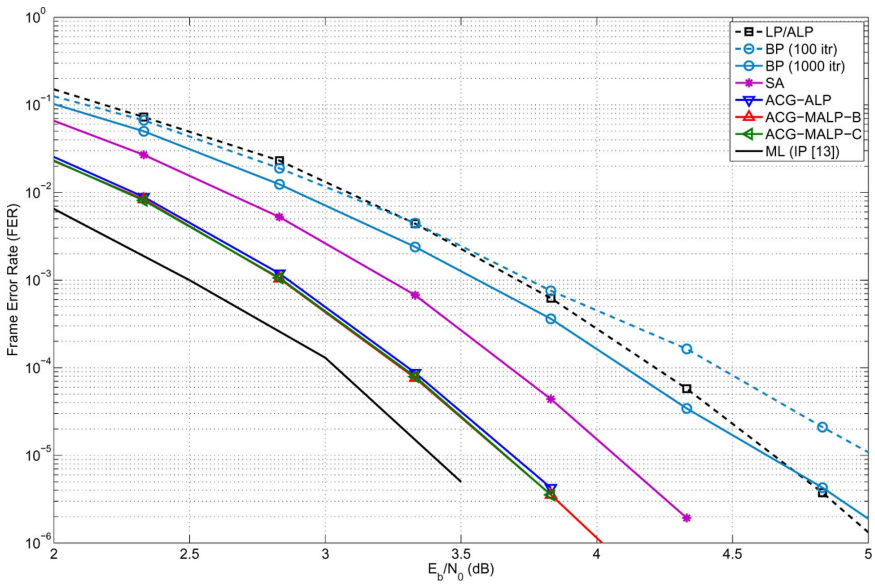

Fig. 3. FER versus E_b/N_0 for (155,64) Tanner LDPC code on the AWGN channel.

solver outputs a solution vector, coordinates with value less than 10^{-6} were rounded to 0 and coordinates with value larger than $1 - 10^{-6}$ were rounded to 1. The rounded values were then used in the cut-search and RPC-generation steps in the decoding algorithms described in previous sections. If such a procedure were not applied, and if, as a result, false cuts were to be produced, the corresponding constraints, when added into the LP problem to be solved in the next step, would leave the solution vector unchanged, causing the decoder to become stuck in an endless loop. We saw no such behavior in our decoder simulations incorporating the prescribed thresholding operations.

Finally, we want to point out that there exist LP solvers, such as *QSopt_ex Rational LP Solver* [22], that produce exact rational solutions to LP instances with rational input. However, such solvers generally have higher computational overhead than their floating-point counterparts. For this reason, we did not use an exact rational LP solver in our empirical studies.

Fig. 1 shows the simulation results for the length-100, regular-(3,4) LDPC code whose frame error rate (FER) performance was also evaluated in [12] and [13]. We can see that the proposed algorithms have a gain of about 2 dB over the original LP and ALP decoders. They also perform significantly better than both the SA and the random-walk algorithm. The figure also shows the results obtained with the box-and-match soft-decision decoding algorithm (BMA) [23], whose FER performance is guaranteed to be within a factor of 1.05 times that of ML decoding. We conclude that the performance gap between the proposed decoders and ML decoding is less than 0.2 dB at an FER of 10^{-5}.

In Fig. 2, we show simulation results for MacKay's length-96, (3,6)-regular LDPC code (the 96.33.964 code from [20]). Again, the proposed ALP-based decoders with ACG demonstrate superior performance to the original LP, BP, and SA decoders over the range of SNRs considered. Table I shows the actual frame error counts for the ACG-ALP decoder, with frame errors classified as either pseudocodewords or incorrect codewords; the ACG-MALP-B and ACG-MALP-C decoder simulations yielded very similar results. We used these counts to obtain a lower bound on ML decoder performance, also shown in the figure, by dividing the number of times the ACG-ALP decoder converged to an incorrect codeword by the total number of frames transmitted. Since the ML certificate property of LP decoding implies that ML decoding would have produced the same incorrect codeword in all of these instances, this ratio represents a lower bound on the FER of the ML decoder. We note that when E_b/N_0 is greater than 4.5 dB, all decoding errors correspond to incorrect codewords, indicating that the ACG-ALP decoder has achieved ML decoding performance for the transmitted frames.

Fig. 3 compares the performance of several different decoders applied to the (3,5)-regular, (155,64) Tanner code, as well as the ML performance curve from [13]. It can be seen that the

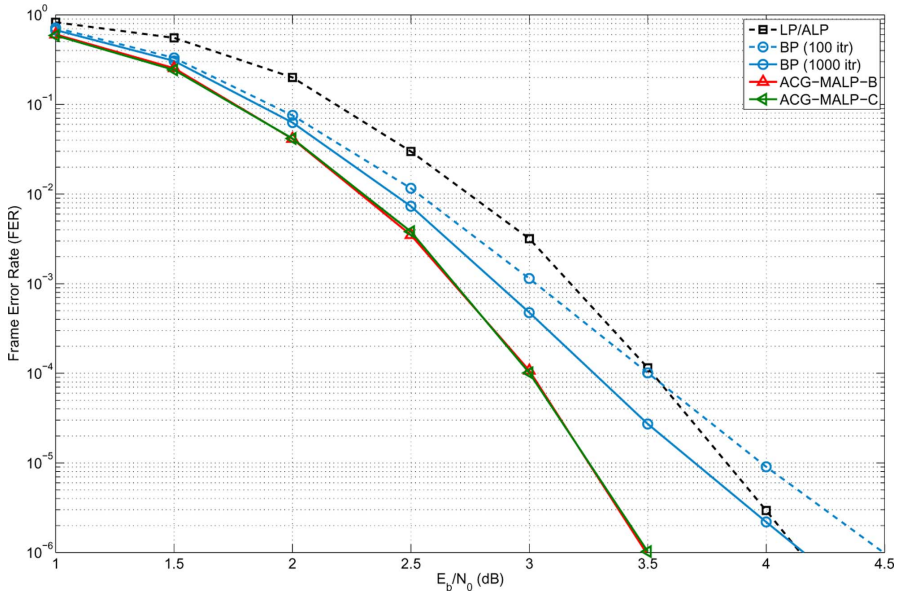

Fig. 4. FER versus E_b/N_0 for MacKay's random (3,6)-regular LDPC code of length 408 on the AWGN channel.

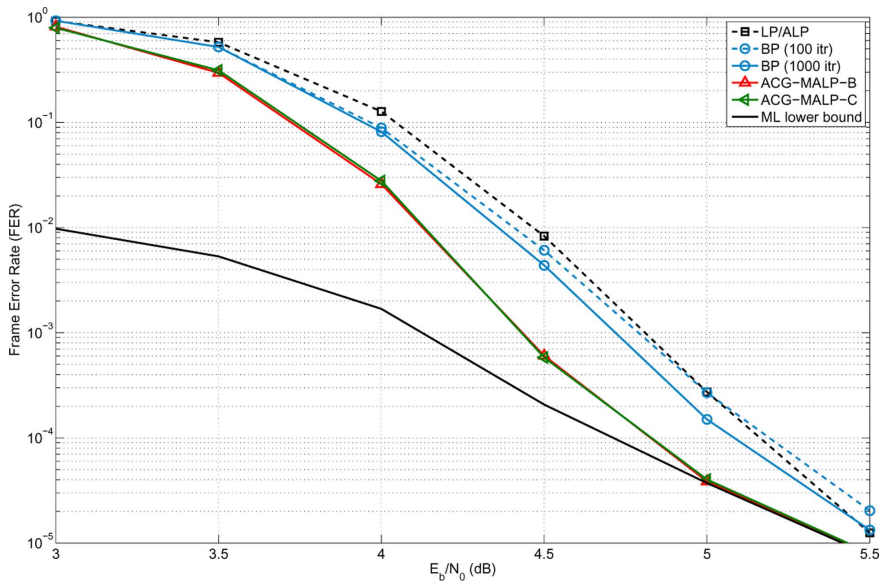

Fig. 5. FER versus E_b/N_0 for MacKay's random (3,27)-regular LDPC code of length 999 on the AWGN channel.

proposed ACG-ALP-based algorithms narrow the 1.25 dB gap between the original LP decoding and ML decoding to approximately 0.25 dB.

We also considered two longer codes: MacKay's rate-$\frac{1}{2}$, random (3,6)-regular LDPC code of length 408 (the 408.33.844 code from [20]) and a rate-0.89 LDPC code of length 999 (the 999.111.3.5543 code from [20]). Because of the increased complexity of the constituent LP problems, we only simulated the ACG-MALP-B and ACG-MALP-C decoders. In Fig. 4, it is confirmed that the proposed decoding algorithms provide significant gain over the original LP decoder and the BP decoder, especially in the high-SNR region. The results for the high-rate LDPC code, as shown in Fig. 5, again show that the proposed decoding algorithms approach ML decoding performance for some codes, where the ML lower bound is obtained using the same technique as in Fig. 2. However, for the code of length 408, we found that the majority of decoding failures corresponded to pseudocodewords, so, in contrast to the case of the length-96 and length-999 MacKay codes discussed previously, the frame error data do not provide a good lower bound on ML decoder performance to use as a benchmark.

Since the observed improvements in ACG-ALP-based decoder performance come from the additional RPC cuts found in each iteration, these decoding algorithms generally require more iterations and/or the solution of larger LP problems in comparison to ALP decoding. In the remainder of this section, we empirically investigate the relative complexity of our proposed algorithms in terms of statistics such as the average number of iterations, the average size of constituent LP problems, and the average number of cuts found in each iteration. All statistical data

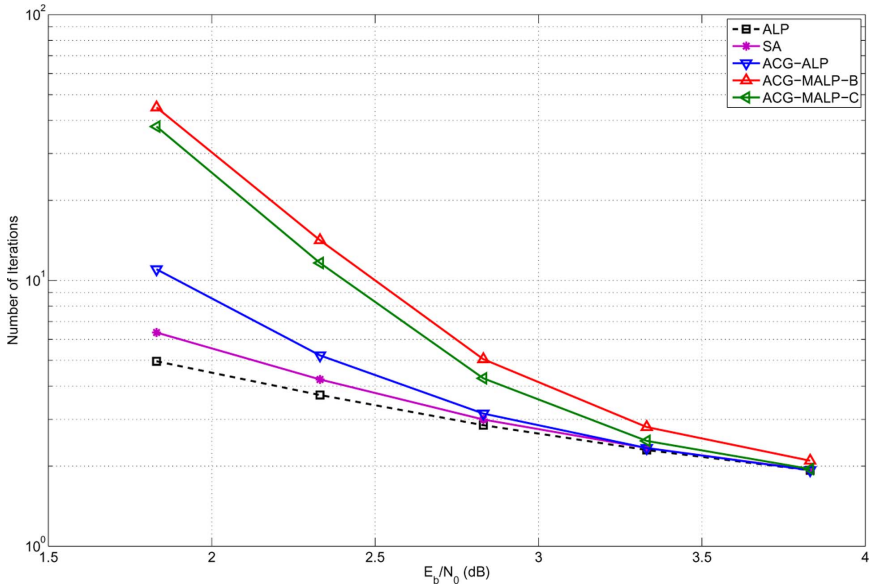

Fig. 6. Average number of iterations for decoding one codeword of (155,64) Tanner LDPC code.

presented here were obtained from simulations of the Tanner (155,64) code on the AWGN channel. We ran all simulations until at least 200 frame errors were counted.

In Fig. 6, we compare the average number of iterations needed, i.e., the average number of LP problems solved, to decode one codeword. Fig. 7(a) compares the average number of constraints in the LP problem of the final iteration that results in either a valid codeword or a pseudocodeword with no more cuts to be found. In Fig. 7(b), we show the average number of cuts found and added into the LP problem in each iteration. Fig. 7(c) and (d) shows the average number of cuts found from the original parity-check matrix $\mathbf{H}$ and from the generated RPCs, respectively.

From Figs. 6 and 7(a), we can see that, as expected, the ACG-ALP decoder takes fewer iterations to decode a codeword on average than the ACG-MALP-B/C decoders, but the ACG-MALP-B/C decoders have fewer constraints in each iteration, including the final iteration. We have observed that the ACG-MALP-B/C decoders require a larger number of iterations to decode than the ACG-ALP decoder, and fewer cuts are added into the constituent LP problems in each iteration on average, as reflected in Fig. 7(b). This is because there are some iterations in which the added constraints had been previously removed. Among all three proposed ACG-based decoding algorithms, we can see that the ACG-ALP decoder has the largest number of constraints in the final iteration and needs the least overall number of iterations to decode, while ACG-MALP-B decoding has the smallest number of constraints but requires the largest number of iterations. The ACG-MALP-C decoder offers a tradeoff between those two: it has fewer constraints than the ACG-ALP decoder and requires fewer iterations than the ACG-MALP-B decoder. If we use the accumulated number of constraints in all iterations to decode one codeword as a criterion to judge the efficiency of these algorithms during simulation, then ACG-MALP-C decoding is more efficient than the other two algorithms in the low and moderate SNR regions, as shown in Table II. Note that the ACG-MALP-B decoder is most efficient at high SNR where the decoding of most codewords succeeds in a few iterations and the chance of a previously removed inactive constraint being added back in later iterations is quite small. Hence, ACG-MALP-B decoding is preferred in the high-SNR region.

Fig. 8 presents an alternative way of comparing the complexity of the decoding algorithms. It shows the average decoding time when we implement the algorithms using C++ code on a desktop PC, with GLPK as the LP solver. The BP decoder is implemented in software with messages represented as double-precision floating-point numbers, and the exact computation of SPA is used, without any simplification or approximation. The BP decoder iterations stop as soon as a codeword is found, or when the maximum allowable number of iterations—here set to 100 and 1000—have been attempted without convergence. The simulation time is averaged over the number of transmitted codewords required for the decoder to fail on 200 codewords.

We observe that the ACG-MALP-B and ACG-MALP-C decoders are both uniformly faster than ACG-ALP over the range of SNR values considered, and, as expected from Table II, ACG-MALP-C decoding is slightly more efficient than ACG-MALP-B decoding in terms of actual running time. Of course, the decoding time depends both on the number of LP problems solved and the size of these LP problems, and the preferred tradeoff depends heavily upon the implementation, particularly the LP solver that is used. Obviously, the improvement in error-rate performance provided by all three ACG-based decoding algorithms over the ALP decoding comes at the cost of increased decoding complexity. As SNR increases, however, the average decoding complexity per codeword of the proposed algorithms approaches that of the ALP decoder. This is because, at higher SNR, the decoders can often successfully decode the received frames without generating RPC cuts.

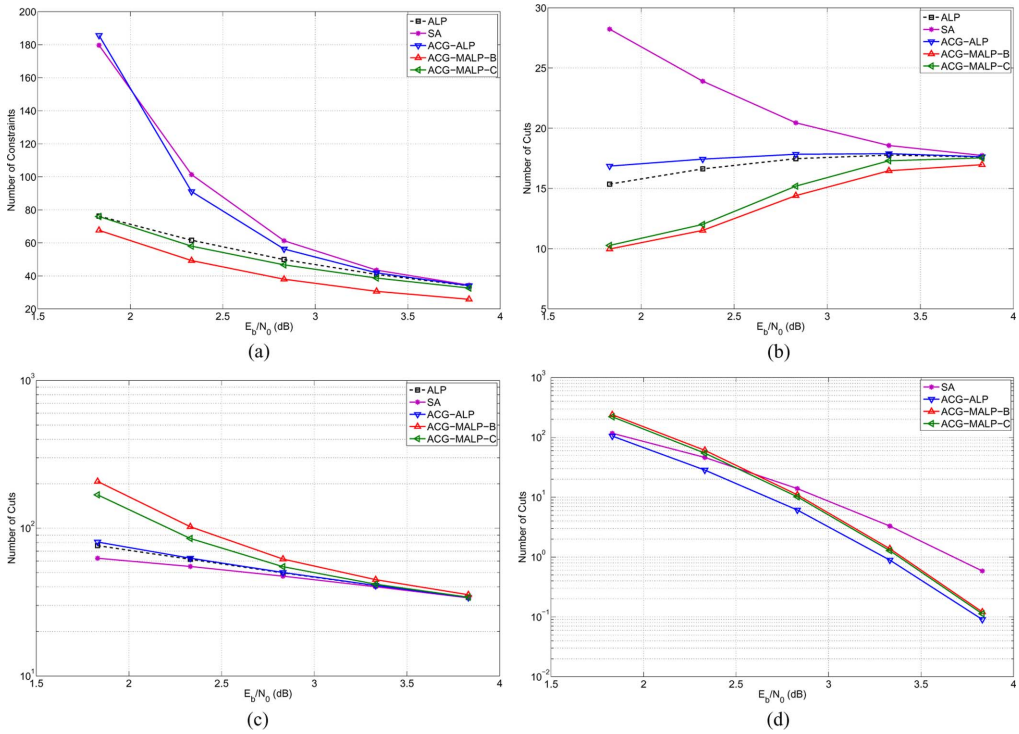

Fig. 7. Average number of constraints/cuts during decoding iterations for decoding one frame of (155,64) Tanner LDPC code. (a) Average number of constraints in final iteration. (b) Average number of cuts found per iteration. (c) Average number of cuts found from $\mathbf{H}$ for decoding one codeword. (d) Average number of cuts found from RPCs for decoding one codeword.

TABLE II
AVERAGE ACCUMULATED NUMBER OF CONSTRAINTS IN ALL ITERATIONS OF DECODING ONE CODEWORD OF (155,64) TANNER CODE ON THE AWGN CHANNEL

E_b/N_0 (dB)	ACG-ALP	ACG-MALP-B	ACG-MALP-C
1.83	5495.8	5223.3	4643.1
2.33	1401.2	1387.3	1217.0
2.83	339.7	326.9	300.9
3.33	111.0	106.4	105.4
3.83	64.3	58.8	62.8

Fig. 6 shows that the ACG-ALP decoder requires, on average, more iterations than the SA decoder. Our observations suggest that this is a result of the fact that the ACG-ALP decoder can continue to generate new RPC cuts after the number of iterations at which the SA decoder can no longer do so and, hence, stops decoding. The simulation data showed that the additional iterations of the ACG-ALP decoder often resulted in a valid codeword, thus contributing to its superiority in performance relative to the SA decoder.

From Fig. 7(b), it can be seen that the ACG-ALP-based decoding algorithms generate, on average, fewer cuts per iteration than the SA decoder. Moreover, as reflected in Fig. 7(c) and (d), the ACG-ALP decoders find more cuts from the original parity-check matrix and generate fewer RPC cuts per codeword. These observations suggest that the CSA is very efficient in finding cuts from a given parity check, while the SA decoder tends to generate RPCs even when there are still some cuts other than the Gomory cuts that can be found from the original parity-check matrix. This accounts for the fact, reflected in Fig. 8, that the SA becomes less efficient as SNR increases, when the original parity-check matrix usually can provide enough cuts to decode a codeword. The effectiveness of our CSA permits the ACG-ALP-based decoders to successfully decode most codewords in the high-SNR region without generating RPCs, resulting in better overall decoder efficiency.

Due to limitations on our computing capability, we have not yet tested our proposed algorithms on LDPC codes of length greater than 1000. We note that, in contrast to [12] and [16], we cannot give an upper bound on the maximum number of iterations required by the ACG-ALP-based decoding algorithms because RPCs and their corresponding parity inequalities are generated adaptively as a function of intermediate pseudocodewords arising during the decoding process. Consequently, even though the decoding of short-to-moderate length LDPC codes was found empirically to converge after an acceptable number of interations, some sort of constraint on the maximum number of iterations allowed may have to be imposed when decoding longer codes. Finally, we point out that the complexity of the algorithm for generating cut-inducing RPCs lies mainly in the Gaussian elimination step, but as applied to binary matrices, this requires only logical operations which can be executed quite efficiently.

VI. CONCLUSION

In this paper, we derived a new necessary condition and a new sufficient condition for a parity-check constraint in a linear block code parity-check matrix to provide a violated parity

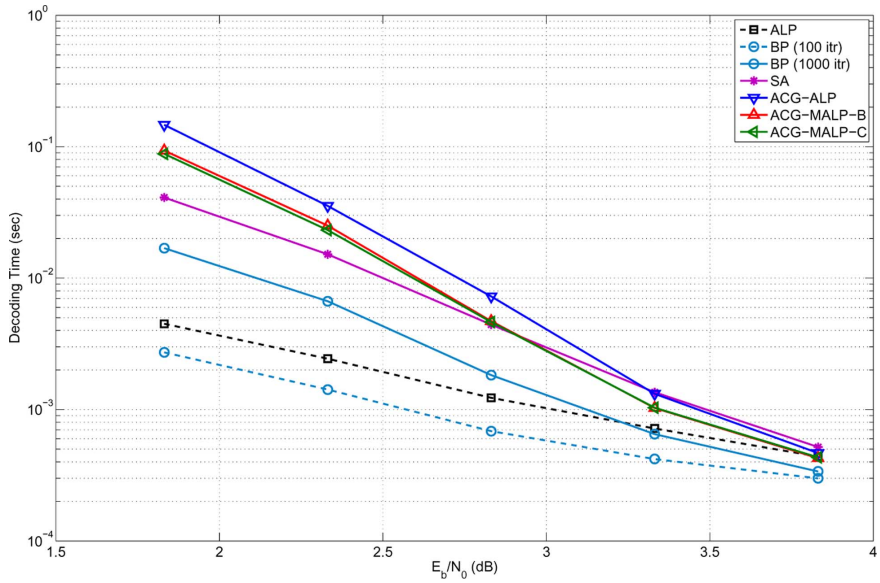

Fig. 8. Average simulation time for decoding one codeword of (155,64) Tanner LDPC code.

inequality, or cut, at a pseudocodeword produced by LP decoding. Using these results, we presented an efficient algorithm to search for such cuts and proposed an effective approach to generating cut-inducing RPCs. The key innovation in the cut-generating approach is a particular transformation of the parity-check matrix used in the definition of the LP decoding problem. By properly reordering the columns of the original parity-check matrix and transforming the resulting matrix into a "partial" reduced row echelon form, we could efficiently identify RPC cuts that were found empirically to significantly improve the LP decoder performance. We combined the new cut-generation technique with three variations of adaptive LP decoding, providing a tradeoff between the number of iterations required and the number of constraints in the constituent LP problems. FER simulation results for several LDPC codes of length up to 999 show that the proposed adaptive cut-generation, adaptive LP (ACG-ALP) decoding algorithms outperform other enhanced LP decoders, such as the SA decoder, and significantly narrow the gap to ML decoding performance for LDPC codes with short-to-moderate block lengths.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, pp. 21–28, Jan. 1962.

[2] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding*, ser. Lecture Notes in Computer Science, C. Boyd, Ed. Heidelberg/Berlin: Springer, 1995, vol. 1025, pp. 100–111.

[3] D. J. C. MacKay and R. M. Neal, "Near Shannon-limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[4] R. McEliece, D. MacKay, and J. Cheng, "Turbo decoding as an instance of Pearl's "belief propagation" algorithm," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 140–152, Feb. 1998.

[5] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[6] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.

[7] P. O. Vontobel and R. Koetter, "Graph-Cover Decoding and Finite-Length Analysis of Message-Passing Iterative Decoding of LDPC Codes," *Comput. Res. Repository*, arxiv.org/abs/cs.IT/0512078.

[8] J. Feldman, "Decoding error-correcting codes via linear programming," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Mass. Inst. Technol., Cambridge, MA, 2003.

[9] K. Yang, X. Wang, and J. Feldman, "A new linear programming approach to decoding linear block codes," *IEEE Trans. Inf. Theory*, vol. 54, no. 3, pp. 1061–1072, Mar. 2008.

[10] M. Chertkov and M. Stepanov, "Pseudo-codeword landscape," in *Proc IEEE Int. Symp. Inf. Theory*, Nice, France, Jun. 2007, pp. 1546–1550.

[11] P. O. Vontobel and R. Koetter, "On low-complexity linear-programming decoding of LDPC codes," *Eur. Trans. Telecommun.*, vol. 18, pp. 509–517, Apr. 2007.

[12] M. H. Taghavi and P. H. Siegel, "Adaptive methods for linear programming decoding," *IEEE Trans. Inf. Theory*, vol. 54, no. 12, pp. 5396–5410, Dec. 2008.

[13] A. Tanatmis, S. Ruzika, H. W. Hamacher, M. Punekar, F. Kienle, and N. Wehn, "A separation algorithm for improved LP-decoding of linear block codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 7, pp. 3277–3289, Jul. 2010.

[14] M. Miwa, T. Wadayama, and I. Takumi, "A cutting-plane method based on redundant rows for improving fractional distance," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 1012–1105, Aug. 2009.

[15] M. Helmling, S. Ruzika, and A. Tanatmis, "Mathematical programming decoding of binary linear codes: Theory and algorithms," *IEEE Trans. Inf. Theory*, vol. 58, no. 7, pp. 4753–4769, Jul. 2012.

[16] M. H. Taghavi, A. Shokrollahi, and P. H. Siegel, "Efficient implementation of linear programming decoding," *IEEE Trans. Inf. Theory*, vol. 57, no. 9, pp. 5960–5982, Sep. 2011.

[17] T. Wadayama, "Interior point decoding for linear vector channels based on convex optimization," *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 4905–4921, Oct. 2010.

[18] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 1990.

[19] GNU linear programming kit [Online]. Available: http://www.gnu.org/software/glpk

[20] D. J. C. MacKay, Encyclopedia of Sparse Graph Codes [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[21] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. Int. Symp. Commun. Theory Appl. (ISCTA)*, Ambleside, U.K., Jul. 2001, pp. 365–369.

[22] (in QSopt Linear Programming Solver) [Online]. Available: http://www2.isye.gatech.edu/~wcook/qsopt/index.html

[23] A. Valembois and M. Fossorier, "Box and match techniques applied to soft decision decoding," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 796–810, May 2004.

Xiaojie Zhang (S'05) received the B.S. degree in Electrical Engineering from Shanghai Jiao Tong University, Shanghai, China, in 2004 and the M.S. degree in Electrical Engineering from Seoul National University, Seoul, Korea, in 2006. From 2006 to 2008, he was a system engineer in Samsung Electronics, Suwon, Korea. He is currently pursuing the Ph.D. degree in the Department of Electrical and Computer Engineering at University of California, San Diego, where he is associated with the Center for Magnetic Recording Research.

His research interests include wireless communication, multiuser scheduling, cross-layer communication system design, and coding theory. His current research interests include error-correction coding, coding theory, and their applications.

Paul H. Siegel (M'82–SM'90–F'97) received the S.B. and Ph.D. degrees in mathematics from the Massachusetts Institute of Technology (MIT), Cambridge, in 1975 and 1979, respectively.

He held a Chaim Weizmann Postdoctoral Fellowship at the Courant Institute, New York University, New York. He was with the IBM Research Division, San Jose, CA, from 1980 to 1995. He joined the faculty of the School of Engineering, University of California, San Diego, in July 1995, where he is currently a Professor of Electrical and Computer Engineering. He is affiliated with the California Institute of Telecommunications and Information Technology, the Center for Wireless Communications, and the Center for Magnetic Recording Research, where he holds an endowed chair and served as Director from 2000 to 2011. His primary research interests lie in the areas of information theory and communications, particularly coding and modulation techniques, with applications to digital data storage and transmission. He holds several patents in the area of coding and detection.

Prof. Siegel was a member of the Board of Governors of the IEEE Information Theory Society from 1991 to 1996 and was re-elected for a three-year term in 2009. He served as Co-Guest Editor of the May 1991 Special Issue on "Coding for Storage Devices" of the IEEE TRANSACTIONS ON INFORMATION THEORY. He served the same TRANSACTIONS as an Associate Editor for Coding Techniques from 1992 to 1995, and as an Editor-in-Chief from July 2001 to July 2004. He was also a Co-Guest Editor of the May/September 2001 two-part issue on "The Turbo Principle: From Theory to Practice" of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS. He was a corecipient, with R. Karabed, of the 1992 IEEE Information Theory Society Paper Award and shared the 1993 IEEE Communications Society Leonard G. Abraham Prize Paper Award with B. Marcus and J. K. Wolf. With J. B. Soriaga and H. D. Pfister, he received the 2007 Best Paper Award in Signal Processing and Coding for Data Storage from the Data Storage Technical Committee of the IEEE Communications Society. He was named a Master Inventor at IBM Research in 1994. He is a member of Phi Beta Kappa and the National Academy of Engineering.