Adaptive Methods for Linear Programming Decoding

Mohammad H. Taghavi N. and Paul H. Siegel, *Fellow, IEEE*

***Abstract*—Detectability of failures of linear programming (LP) decoding and the potential for improvement by adding new constraints motivate the use of an adaptive approach in selecting the constraints for the underlying LP problem. In this paper, we make a first step in studying this method, and show that by starting from a simple LP problem and adaptively adding the necessary constraints, the complexity of LP decoding can be significantly reduced. In particular, we observe that with adaptive LP decoding, the sizes of the LP problems that need to be solved become practically independent of the density of the parity-check matrix. We further show that adaptively adding extra constraints, such as constraints based on redundant parity checks, can provide large gains in the performance.**

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

I. Introduction

LINEAR programming (LP) decoding, as an approximation to maximum-likelihood (ML) decoding, was proposed by Feldman, Wainwright, and Karger [1]. Many observations suggest similarities between the performance of LP and iterative message-passing decoding methods. For example, we know that the existence of low-weight pseudocodewords degrades the performance of both types of decoders [1]–[3]. Moreover, the sum–product message-passing algorithm can be interpreted as a minimization of a nonlinear function, known as Bethe free energy, over the same feasible region as LP decoding [4], [5]. Therefore, it is reasonable to try to exploit the simpler geometrical structure of LP decoding to make predictions about the performance of message-passing decoding algorithms.

On the other hand, there are differences between these two decoding approaches. For instance, given a low-density parity-check (LDPC) code, we know that adding redundant parity checks that are satisfied by all the codewords cannot degrade the performance of LP decoding, while with message-passing algorithms, these parity checks may have a negative effect by introducing short cycles in the corresponding Tanner graph. This property of LP decoding allows performance improvement by tightening the relaxation. Another characteristic of LP decoding—the *ML certificate property*—is that its failure to find an ML codeword is always detectable. More specifically, the decoder always gives either an ML codeword or a nonintegral pseudocodeword as the solution.

These two properties motivate the use of an adaptive cutting-plane approach in LP decoding which can be summarized as follows: Given a set of constraints that describe a code, start the LP decoding with a few of them, then sequentially and adaptively add more of the constraints to the problem until either an ML codeword is found or no further "useful" constraint exists. The goal of this paper is to explore the potential of this idea for improving the performance of LP decoding.

In the original formulation of LP decoding, the number of constraints per check node is exponential in the check node degrees. While for LDPC codes this number is independent of the code length n, even for small check degrees this could result in a very large linear program. Furthermore, for high-density codes, where the degrees grow with n, the size of the LP decoding problem will be exponential in n. In [1], Feldman *et al.* provide an alternative equivalent representation of the LP relaxation, where the total number of constraints is $O(n^3)$ for codes of high check node degrees. However, for practical purposes, this formulation is still not very feasible.

In this paper, we show that by incorporating adaptivity into the LP decoding procedure, we can achieve with a small number of constraints the same error-rate performance as that obtained when standard LP decoding is applied to a relaxation defined by a much larger number of constraints. In particular, we observe that the proposed adaptive LP decoding method generally converges with a (small) constant number of constraints per check node that does not appear to be dependent upon the underlying code's degree distribution. This property makes it feasible to apply adaptive LP decoding to graph codes of arbitrary degree distribution, and reduces the decoding time by orders of magnitude relative to the standard implementation, even for relatively small check node degrees. Recently, Draper, Yedidia, and Wang [6] used this algorithm in a mixed-integer optimization setting for ML decoding of moderate-sized LDPC codes.

In related work, Chertkov and Stepanov [7], and Yang, Wang, and Feldman [8], independently proposed a new representation of the LP decoding problem, where each high-degree check node is decomposed into a number of degree-3 check nodes and some auxiliary variable nodes. Using this scheme, the number of variables and constraints of the LP grows linearly with the check degrees, and how the overall decoding complexity scales will depend on the underlying LP solver. A different line of work in this direction has been to apply iterative methods based on message-passing, instead of general LP solvers, to perform the optimization for LP decoding. An important result in this area is the tree-reweighted message-passing algorithm by Wainwright, Jaakkola, and Willsky [5]. Furthermore, Vontobel and Koetter

Manuscript received March 24, 2007; revised May 20, 2008. Current version published November 21, 2008. This work was supported by the Center for Magnetic Recording Research at University of California, San Diego and from the National Science Foundation under Grants CCF-0514859 and CCF-0829865. The material in this paper was presented in part at the IEEE International Symposium on Information Theory, Seattle, WA, July 2006.

M. H. Taghavi N. was with the Department of Electrical and Computer Engineering, and the Center for Magnetic Recording Research, University of California, San Diego, La Jolla, CA 92093-0401 USA. He is now with Qualcomm Inc., San Diego, CA 92121 USA (e-mail: mtaghavi@ucsd.edu).

P. H. Siegel is with the Department of Electrical and Computer Engineering, and the Center for Magnetic Recording Research, University of California, San Diego, La Jolla, CA 92093-0401 USA (e-mail: psiegel@ucsd.edu).

Communicated by H.-A. Loeliger, Associate Editor for Coding Techniques.

Digital Object Identifier 10.1109/TIT.2008.2006384

proposed an approximation to LP decoding by applying an iterative min-sum algorithm-type technique to the dual of the LP decoding problem [9].

In the second part of this paper, we show how the cutting-plane concept can be used to improve the error-rate performance of LP decoding. In this scheme, once the adaptive LP decoding fails to find an integral (i.e., the ML) codeword, we iteratively search for and add a number of cuts, i.e., new constraints that cut the current nonintegral solution from the feasible space, and then we solve the new LP problem. While there is a wide range of general-purpose techniques for finding cutting planes, in this work we focus on cuts that are obtained based on redundant parity checks. We present some results on the properties of these cuts, and demonstrate how considerable gains can be obtained by using this technique. Other approaches toward improving the performance of LP decoding include facet guessing, proposed by Dimakis and Wainwright [10], and the application of loop calculus, suggested by Chertkov and Chernyak [11].

Along the way, we prove several general properties of LP relaxations of ML decoding that shed light upon the performance of LP and iterative decoding algorithms.

The rest of this paper is organized as follows. In Section II, we review Feldman *et al.*'s LP decoding. In Section III, we introduce and analyze the adaptive LP decoding algorithm to solve the original LP problem more efficiently. In Section IV, we study how adaptively imposing additional constraints can improve the LP decoder performance. Section V concludes the paper.

II. LP Relaxation of ML Decoding

Consider a binary linear code $\mathfrak{C}$ of length n. If a codeword $\underline{y} \in \mathfrak{C}$ is transmitted through a memoryless binary-input output-symmetric (MBIOS) channel, the ML codeword given the received vector $\underline{r} \in \mathbb{R}^n$ is the solution to the optimization problem

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

where $\underline{\gamma}$ is the vector of log-likelihood ratios defined as

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

As an approximation to ML decoding, Feldman *et al.* proposed a relaxed version of this problem by first considering the convex hull of the local codewords defined by each row of the parity-check matrix, and then intersecting them to obtain what Koetter *et al.* [3] called the *fundamental polytope*, $\mathfrak{P}$. This polytope has a number of integral and nonintegral vertices, but the integral vertices exactly correspond to the codewords of $\mathfrak{C}$. Therefore, the LP relaxation has the *ML certificate property*, i.e., whenever LP decoding gives an integral solution, it is guaranteed to be an ML codeword.

In Feldman *et al.*'s formulation of the decoding problem, constraints are derived from a parity-check matrix as follows. For each row $j = 1, \ldots, m$ of the parity-check matrix, define the neighborhood set $N(j) \subseteq \{1, 2, \ldots, n\}$ of the corresponding check node in the Tanner graph to be the variable nodes that are directly connected to it. (For convenience, we often identify check nodes and variable nodes with their respective index sets $j = 1, \ldots, m$ and $i = 1, \ldots, n$.) Then, for $j = 1, \ldots, m$, the LP formulation includes all of the following constraints:

$$\sum_{i \in V} x_i - \sum_{i \in N(j) \backslash V} x_i \leq |V| - 1, \ \forall\, V \subseteq N(j) \text{ s. t. } |V| \text{ is odd}. \tag{3}$$

We refer to the constraints of this form as *parity inequalities*. If the variables x_i are zeros and ones, these constraints will be equivalent to the original binary parity-check constraints. To see this, note that if V is a subset of $N(j)$, with $|V|$ odd, and the corresponding parity inequality fails to hold, then all variable nodes in V must have the value 1, while those in $N(j) \backslash V$ must have the value 0. This implies that the corresponding vector $\underline{x}$ does not satisfy parity check j. Conversely, if parity check j fails to hold, there must be a subset of variable nodes $V \subseteq N(j)$ of odd size such that all nodes in V have the value 1 and all those in $N(j) \backslash V$ have the value 0. Clearly, the corresponding parity inequality would be violated. Now, given this equivalence, we relax the LP problem by replacing each binary constraint $x_i \in \{0, 1\}$ by a *box constraint* $0 \leq x_i \leq 1$.

III. Adaptive LP Decoding

As any odd-sized subset V of the neighborhood $N(j)$ of a check node j introduces a unique parity inequality, there are $2^{d_c - 1}$ constraints corresponding to each check node of degree d_c in the original formulation of LP decoding. Therefore, the total number of constraints, and hence the complexity of the problem, is exponential in terms of the maximum check node degree $d_c^{\max}$. This becomes more significant in a high-density code where $d_c^{\max}$ increases with the code length n. As discussed earlier, Feldman *et al.* proposed an equivalent formulation that has a problem size of $O(n^3)$, although for low-density codes it has a problem size larger than the original formulation.[1]

In this section, we show that the LP relaxation explained in Section II has some properties that allow us to solve the optimization problem by using a much smaller number of constraints.

A. Properties of the Relaxation Constraints

Definition 1: Given a constraint of the form

$$\underline{a}_i^T \underline{x} \leq b_i \tag{4}$$

and a vector $\underline{x}_0 \in \mathbb{R}^n$, we call (4) an *active constraint* at $\underline{x}_0$ if

$$\underline{a}_i^T \underline{x}_0 = b_i \tag{5}$$

and a *violated constraint* or, equivalently, a *cut* at $\underline{x}_0$ if

$$\underline{a}_i^T \underline{x}_0 > b_i. \tag{6}$$

A constraint that generates a cut at point $\underline{x} \in [0, 1]^n$ corresponds to a subset $V \subseteq N(j)$ of odd cardinality such that

$$\sum_{i \in V} x_i - \sum_{i \in N(j) \backslash V} x_i > |V| - 1. \tag{7}$$

[1]More recently, an alternative polytope was proposed in [7] and [8] with a number of variables and constraints that grows linearly with the code length and the maximum check node degree.

This condition implies that

$$|V|-1<\sum_{i\in V}x_i\le|V| \tag{8}$$

and

$$0\le\sum_{i\in N(j)\backslash V}x_i<x_\ell,\quad\forall\ell\in V. \tag{9}$$

The following theorem reveals the special property of the relaxed parity inequalities (3) that at most one of the constraints introduced by each check node can be a cut.

Theorem 1: If at any given point $\underline{x}\in[0,1]^n$, one of the parity inequalities introduced by a check node j is a cut, the rest of the parity inequalities from this check node are satisfied with strict inequality.

Proof: For a check node j, we first rewrite (3) as

$$\sum_{i\in V}(1-x_i)+\sum_{i\in N(j)\backslash V}x_i\ge 1,\ \forall\, V\subseteq N(j)\text{ s. t. }|V|\text{ is odd}. \tag{10}$$

Let $\underline{x}_{N(j)}$ denote the vector composed of the elements of $\underline{x}$ with indices in $N(j)$, and by $\underline{I}_V$ the indicator vector of the subset V of $N(j)$, i.e., a vector of length $N(j)$ with the kth element being 1 if $k\in V$, and 0 otherwise. Then, since $\underline{x}\in[0,1]^n$, (10) will be equivalent to

$$d_{L_1}\left(\underline{x}_{N(j)},\underline{I}_V\right)\ge 1,\ \forall\, V\subseteq N(j)\ \text{ s. t. }|V|\text{ is odd} \tag{11}$$

where $d_{L_1}(\underline{a},\underline{b})=\sum_k|a_k-b_k|$ is the L_1-distance between $\underline{a}$ and $\underline{b}$.

Consider a check node with neighborhood $N(j)\subseteq\{1,2,\dots,n\}$ and two distinct subsets $V_1\subseteq N(j)$ and $V_2\subseteq N(j)$ of odd sizes $|V_1|$ and $|V_2|$, respectively, such that V_1 introduces a cut at point $\underline{x}$. This means that the L_1-distance between $\underline{x}_{N(j)}$ and $\underline{I}_{V1}$ is less than 1. In addition, since $\underline{I}_{V1}$ and $\underline{I}_{V2}$ are indicator vectors of two distinct odd subsets, their L_1-distance should be at least 2. Hence, using the triangle inequality, we will have

$$\begin{aligned}2&\le d_{L_1}(\underline{I}_{V1},\underline{I}_{V2})\\&\le d_{L_1}(\underline{x}_{N(j)},\underline{I}_{V1})+d_{L_1}(\underline{x}_{N(j)},\underline{I}_{V2})\\&<1+d_{L_1}(\underline{x}_{N(j)},\underline{I}_{V2}).\end{aligned} \tag{12}$$

Therefore, $d_{L_1}(\underline{x}_{N(j)},\underline{I}_{V2})>1$, which means that the parity inequality introduced by V_2 is satisfied with strict inequality. □

Given an (n,k) linear code with $m=n-k$ parity checks, a natural question is how we can find all the cuts defined by the LP relaxation at any given point $\underline{x}\in[0,1]^n$. Referring to (9), we see that for any check node and any odd-sized subset V of its neighborhood $N(j)$ that introduces a cut, the variable nodes in V have the largest values among all of the nodes in $N(j)$. Therefore, sorting the elements of $\underline{x}$ can simplify the process of searching for a cut. This observation is reflected in Algorithm 1 below.

Consider a check node j with neighborhood $N(j)$. Without loss of generality, assume that variable nodes in $N(j)$ have indices $1,2,\dots,|N(j)|$, and that their values are sorted, such that $x_1\ge x_2\ge\cdots\ge x_{|N(j)|}$. The following algorithm provides an efficient way to find the unique cut generated by this check node at $\underline{x}$, if a cut exists.

Algorithm 1:

Step 1: Set $v=1, V=\{1\}$ and $V^c\triangleq N(j)\backslash V=\{2,3,\dots,|N(j)|\}$.

Step 2: Check the constraint (3). If it is violated, we have found the cut. Exit.

Step 3: Set $v=v+2$. If $v\le|N(j)|$, move x_{v-1} and x_v (the two largest members of V^c) from V^c to V.

Step 4: If $v\le|N(j)|$ and (8) is satisfied, go to Step 2; otherwise, the check node does not provide a cut at x.

Regarding Step 4, note that v equals the cardinality of the set $V\subseteq N(j)$, and hence it should remain less than or equal to $|N(j)|$. In addition, the failure of condition (8), a necessary condition for having a cut, provides a definitive termination criterion for Algorithm 1. For, if at some iteration the condition fails to hold for the set V, i.e., $|V|-1\ge\sum_{i\in V}x_i$, it would necessarily fail for the set V considered at any subsequent iteration.

If redundant calculations are avoided in calculating the sums in (3), this algorithm can find the cut generated by the check node, if it exists, in $O(d_c)$ time, where $d_c=|N(j)|$ is the degree of the check node. Repeating the procedure for each check node, and incurring $O(n\log n)$ complexity for sorting $\underline{x}$, the time required to find all the cuts at point $\underline{x}$ becomes $O(md_c^{\max}+n\log n)$.[2]

B. The Adaptive Procedure

The fundamental polytope for a parity-check code is defined by a large number of constraints (hyperplanes), and a linear programming solver finds the vertex (pseudocodeword) of this polytope that minimizes the objective function. For example, the Simplex algorithm starts from an initial vertex and visits different vertices of the polytope by traveling along the edges, which is called *pivoting*, until it finds the optimum vertex. The time required to find the solution equals the product of the number of vertices that have been visited and the average processing time at each pivot step, and these, in turn, are determined by the number and properties of constraints and variables in the problem. If we can reduce the size of the problem without changing the optimum vertex, the amount of processing at each pivot, which usually involves matrix computations, will decrease significantly. Furthermore, reducing the number of vertices that are visited in the intermediate iterations can reduce the complexity of the algorithm. In the adaptive LP decoding scheme, we implement this idea using a cutting-plane approach. We run the LP solver with a minimal number of constraints to ensure boundedness of the solution, and depending on the LP solution, we add only the "useful constraints" that cut the current solution from the feasible region. This procedure is repeated until no further cut exists, which means that the solution is a vertex of the fundamental polytope.

[2]For LDPC codes, it is better to sort the neighbors of each check node separately, so the total complexity becomes

$$O(md_c^{\max}+md_c^{\max}\log d_c^{\max})=O(md_c^{\max}\log d_c^{\max}).$$

To start the procedure, we need at least n constraints to determine a vertex that can become the solution of the first iteration. Recalling the box constraints $0 \leq x_i \leq 1$, we add for each i exactly one of the inequalities implied by these constraints. The choice depends upon whether increasing x_i leads to an increase or decrease in the objective function, to ensure that the solution to the initial LP is bounded. Specifically, for each $i \in \{1, 2, \ldots, n\}$, we introduce the initial constraint

$$\begin{cases} 0 \leq x_i, & \text{if } \gamma_i \geq 0 \\ x_i \leq 1, & \text{if } \gamma_i < 0. \end{cases} \tag{13}$$

Note that the optimum (and only) vertex satisfying this initial problem corresponds to the result of an (uncoded) bit-wise, hard decision based on the received vector.

The following algorithm describes the adaptive LP decoding procedure.

Algorithm 2:

Step 1: Form the initial problem with the constraints from (13); set $k = 0$.

Step 2: Run the LP solver to obtain the current solution $\underline{x}_k$.

Step 3: If any of the box constraints are violated at $\underline{x}_k$, add them to the problem and go to Step 2.

Step 4: If one or more of the parity inequalities are violated at $\underline{x}_k$, add them to the problem, set $k = k + 1$, and go to Step 2; else, we have found the solution.

Although in Step 3 of the algorithm we search the box constraints for cuts, in practice we have never observed a case where a box constraint is violated throughout the algorithm. We conjecture that by initially including only one inequality from the box constraints of each variable according to (13), there is no need to check for any violated box constraints in Step 3 of Algorithm 2.[3] Hence, through the rest of this section, an "iteration" of Algorithm 2 will refer to an execution of Step 4.

Lemma 1: If no cut is found after any iteration of Algorithm 2, the current solution represents the solution of the standard LP decoding problem incorporating all of the relaxation constraints given in Section II.

Proof: At any intermediate step of the algorithm, the space of feasible points with respect to the current constraints contains the fundamental polytope $\mathfrak{P}$, as these constraints are all among the original constraints used to define $\mathfrak{P}$. If at any iteration, no cut is found, we conclude that all the original constraints given by (3) are satisfied by the current solution $\underline{x}$, which means that this point is in $\mathfrak{P}$. Hence, since $\underline{x}$ has the lowest cost in a space that contains $\mathfrak{P}$, it is also the optimum point in $\mathfrak{P}$. □

To further speed up the algorithm, we can use a "warm start" after adding a number of constraints at each iteration. In other words, since the intermediate solutions of the adaptive algorithm converge to the solution of the original LP problem, we can use the solution of each iteration as a starting point for the next iteration. While there are warm start strategies for both the Simplex and interior-point algorithms, using warm starts has been observed to be more effective for the Simplex algorithm [12].

[3]*Note added in proof:* Indeed, in a recent work to be published, we have been able to establish a proof of this conjecture.

In the Simplex algorithm, since the next solution will usually be close to the current solution, the number of iterations (pivots), and therefore, the overall running time, is expected to decrease. However, each of these warm starts is a basic infeasible point for the subsequent problem, since it will not satisfy the newly added constraints. As a result, a primal–dual implementation of the Simplex method will first take a few steps to move back into the feasible region. In Subsection III-D, we will discuss in more detail the effect of using warm starts on the speed of the algorithm.

C. A Bound on the Complexity

Theorem 2: The adaptive algorithm (Algorithm 2) converges after at most n iterations.

Proof: The solution produced by the algorithm is a vertex $\underline{x}_f$ of the feasible space determined by the initial constraints along with those added by the successive iterations of the cut-finding procedure. Therefore, we can find n such constraints

$$\kappa_i : \underline{\alpha}_i^T \underline{x} \leq \beta_i, \quad i = 1, 2, \ldots n$$

that are linearly independent and whose corresponding hyperplanes uniquely determine this vertex. This means that if we set up an LP problem with only those n constraints, the optimum point will be $\underline{x}_f$. Now, consider the kth intermediate solution, $\underline{x}_k$, that is cut from the feasible space at the end of the kth iteration of Algorithm 2. At least one of the constraints, $\kappa_1, \ldots, \kappa_n$, should be violated by $\underline{x}_k$; otherwise, since $\underline{x}_k$ has a lower cost than $\underline{x}_f$, $\underline{x}_k$ would be the solution of the LP with these n constraints. But we know that the cuts added at the kth iteration are all the possible constraints that are violated at $\underline{x}_k$. Consequently, at least one of the cuts added at each iteration should be among $\{\kappa_i : i = 1, 2, \ldots, n\}$; hence, the number of iterations is at most n. □

Remark 1: The adaptive procedure and convergence result can be generalized to any LP problem defined by a fixed set of constraints. In general, however, there may not be an analog of Theorem 1 to facilitate the search for cut constraints.

Remark 2: If, for a given code of length n, the adaptive algorithm converges with at most $q < n$ final parity inequalities, then each pseudocodeword of this LP relaxation should have at least $n - q$ integer elements. To see this, note that each pseudocodeword corresponds to the intersection of at least n active constraints. If the problem has at most q parity inequalities, then at least $n - q$ constraints of the form $x_i \geq 0$ or $x_i \leq 1$ should be active at each pseudocodeword, which means that at least $n - q$ positions of the pseudocodeword are integer-valued.

Corollary 1: The final application of the LP solver in the adaptive decoding algorithm uses at most $n(m+2)$ constraints.

Proof: There are $2n$ box constraints, and, according to Theorem 1, at each iteration no more than m new parity inequalities are added. Since there are at most n iterations, the final number of constraints is at most $n(m + 2)$. □

For very low-density codes, where $2^{d_c^{\max}} \leq m$, the above bound does not imply a complexity improvement over the

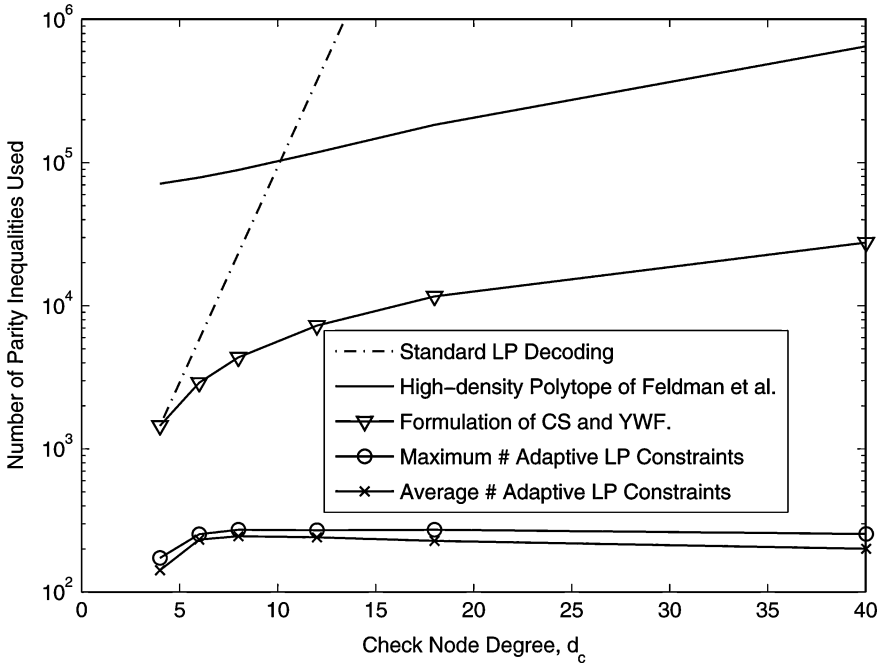

Fig. 1. The average and maximum number of parity inequalities used versus check node degree d_c, for fixed length $n = 360$ and rate $R = \frac{1}{2}$.

original formulation of LP decoding. However, for high-density codes of fixed rate, as the code length and the number of parity checks increase proportionately, this bound guarantees convergence with $O(n^2)$ constraints, whereas the standard LP relaxation requires a number of constraints that is exponential in n, and the high-density code polytope representation given in [1, Appendix II] involves $O(n^3)$ variables and constraints.[4] Moreover, Theorem 2 does not make use of the properties of the decoding problem, in particular, its sparsity. Therefore, especially in the case of low-density codes, we do not expect this bound to be tight.

D. Numerical Results

To empirically investigate the complexity reduction due to the adaptive approach for LP decoding, we performed simulations over random regular LDPC codes of various lengths, degrees, and rates on the additive white Gaussian noise (AWGN) channel. In all the simulations in this paper, the signal-to-noise ratio (SNR) is defined as the ratio of the variance of the transmitted discrete-time signal to the variance of the noise sample.

We first demonstrate the simulation results for adaptive LP decoding at the low-SNR value of -1.0 dB, since in the high-SNR regime the received vector is likely to be close to a codeword, in which case the algorithm may converge more rapidly, rather than demonstrating its worst case behavior. Afterwards, we study how the behavior of this decoder changes as the SNR grows.

1) Low-SNR Regime: For each point in the following figures, 400 codeword transmissions were simulated, using a randomly generated, but fixed, regular LDPC code, with its 4-cycles removed.

In the first scenario, we studied the effect of changing the check node degree d_c from 4 to 40 while keeping the code length at $n = 360$ and the rate at $R = \frac{1}{2}$. The average (resp., maximum) number of iterations required to converge started from around 14.5 (resp., 30) for the $(2, 4)$ code, and decreased monotonically down to 5.9 (resp., 9) for the $(20, 40)$ code. The average and maximum number of parity inequalities used in the final iteration of the algorithm are plotted in Fig. 1. We see that both the average and the maximum values are almost constant, and remain below 270 for all the values of d_c. The only exception to this was the first point, corresponding to the $(2, 4)$ code, which, on average, has a number of parity inequalities significantly smaller, and a number of iterations significantly larger than those of the other codes. For comparison, the total number of parity inequalities required in the standard representation of the LP decoding problem in the high-density code polytope representation of Feldman *et al.* [1], as well as in the polytope representation of Chertkov–Stepanov (CS) [7] and Yang–Wang–Feldman (YWF) [8], are also included in the figure. The decrease in the number of required constraints translates to a significant advantage for the adaptive algorithm in terms of the running time.

In the second case, we studied random $(3, 6)$-regular codes of lengths $n = 30$ to $n = 1920$. For all values of n, the average (resp., maximum) number of required iterations remained between 5 and 11 (resp., 10 and 16). The average and maximum number of parity inequalities used in the final iteration are plotted versus n in Fig. 2. We observe that the number of parity inequalities is generally between $0.6n$ and $0.7n$.

In the third experiment, we investigated the effect of the rate of the code on the performance of the algorithm. Fig. 3 shows

[4]This $O(n^2)$ bound is comparable to the recent results in [7] and [8], where a formulation was proposed for LP decoding requiring only $O(n^2)$ variables and constraints for high-density codes.

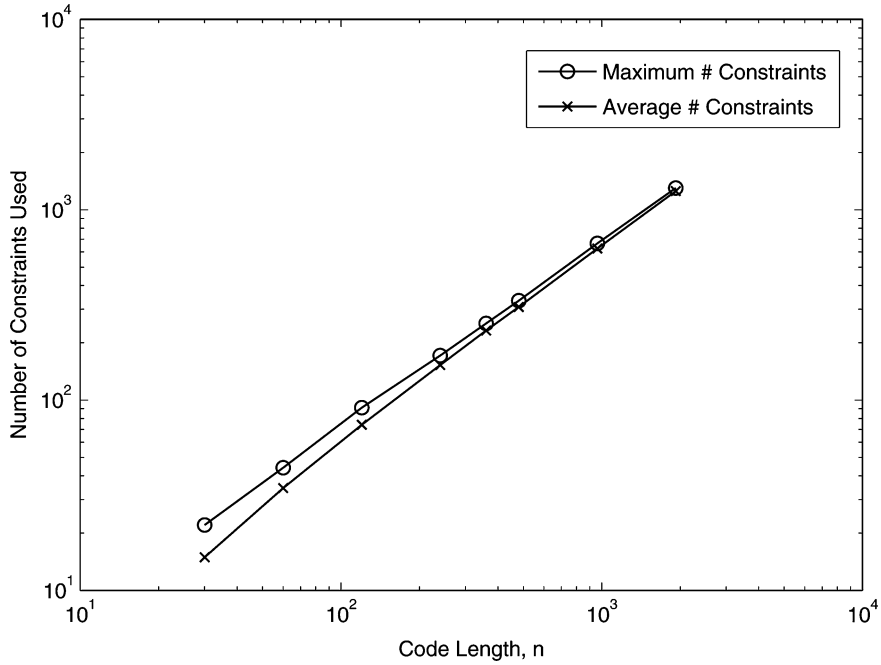

Fig. 2. The average and maximum number of parity inequalities used versus block length n, for fixed rate $R = \frac{1}{2}$ and check node degree $d_c = 6$.

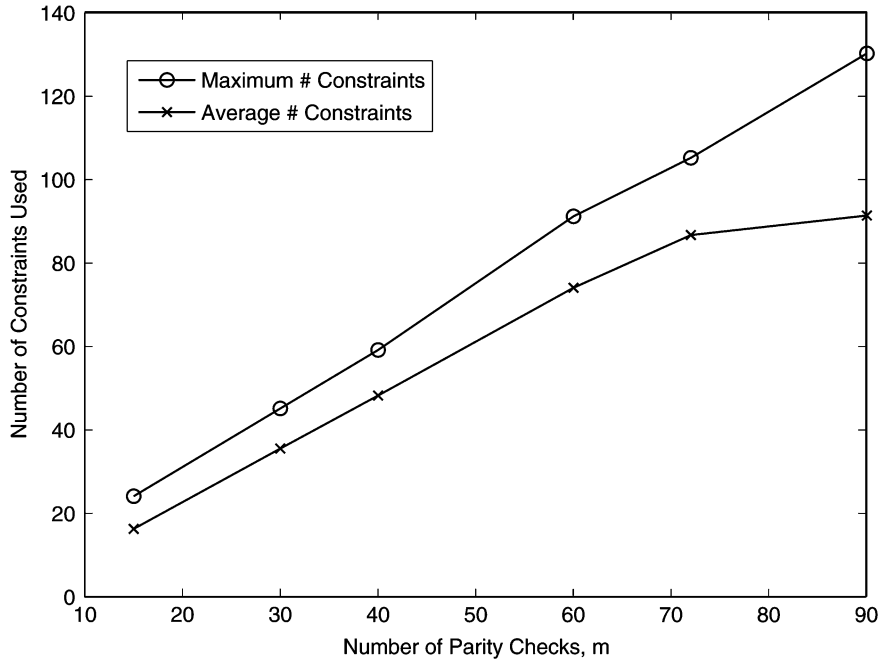

Fig. 3. The average and maximum number of parity inequalities used versus the number of parity checks m, for $n = 120$ and $d_v = 3$.

the average and maximum number of parity inequalities used in the final iteration where the block length is $n = 120$ and the number of parity checks m, increases from 15 to 90. The variable node degree is fixed at $d_v = 3$. We see that the average and maximum number of parity inequalities used are, respectively, in the ranges $1.1m$ to $1.2m$ and $1.4m$ to $1.6m$ for most values of m. There is a relatively large drop in the average number for $m = 90$ with respect to the linear curve. This can be explained by the fact that at this value of m, which corresponds to a $(3, 4)$-regular code, the rate of failure of LP decoding was less than 0.5 at -1.0 dB, whereas for all the other values of m, this rate was close to 1. As we will see later in this subsection, this different behavior at a lower error rate is consistent with the observation that a successful decoding typically requires fewer iterations, and as a result, fewer constraints, than a decoding resulting in a fractional output.

Finally, in Fig. 4, we compare the average decoding time of different algorithms at the low SNR of -1.0 dB. It is important to note that the running times of the LP-based techniques strongly depend on the underlying LP solver. In this work, we have used the Simplex algorithm implementation of the open-source GNU Linear Programming Kit (GLPK [13]) for solving the LPs. The numerical results demonstrate that the adaptive algorithm significantly reduces the gap between the speed of stan-

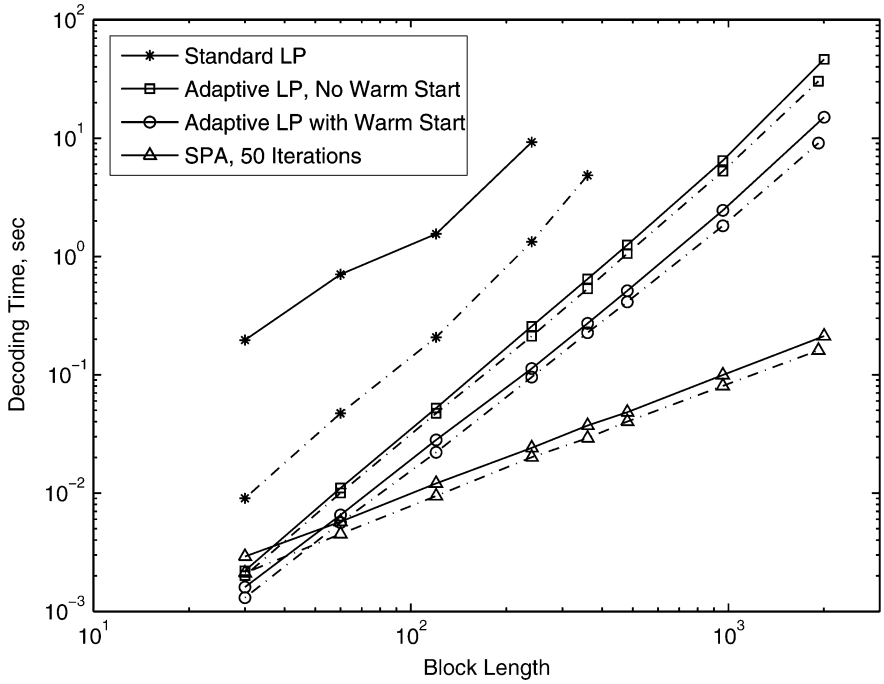

Fig. 4. The average decoding time versus the length of the code for $(3, 6)$-regular LDPC codes (dashed lines) and $(4, 8)$-regular LDPC codes (solid lines) at SNR $= -1.0$ dB.

dard LP decoding and that of the sum–product message-passing algorithm. Comparing the results for the $(3,6)$-regular codes (dot-dashed lines) and the $(4,8)$-regular codes (solid lines) further shows that while the decoding time for the standard LP increases very rapidly with the check node degree of the code, the adaptive technique is not significantly affected by the change in the check node degree.

Other simulations we performed indicate that the decoding time of the adaptive algorithm does not change significantly even if the code has a high-density parity-check matrix. This result can be explained by two factors. First, Fig. 1 shows that the number of parity inequalities used in the algorithm does not change substantially with the check node degree of the code. Second, while having a smaller check degree makes the matrix of constraints sparser, the LP solver that we are using does not benefit from this sparsity. A similar behavior was also observed when we tried both the Simplex and interior-point implementations of a commercial LP solver, MOSEK [14], instead of GLPK. Therefore, an interesting question for future investigation is whether, by designing a special LP solver that can effectively take advantage of the sparsity of this problem, the time complexities of the LP-based techniques can become even closer to those of the message-passing techniques.

Fig. 4 also shows the average decoding time when warm starts are used in the iterations of the adaptive decoding algorithm. We can see that warm starts slightly decrease the slope of the decoding-time curve when plotted against the logarithm of the block length. This translates into approximately a factor of 3 improvement in the decoding time at a block length of 1000.

In all our simulations, we have observed that the adaptive LP decoding algorithm performs much faster than is guaranteed by Theorem 2. In particular, the average number of iterations of Algorithm 2, as well as the average number of parity inequalities per check node used in this algorithm, do not appear to change significantly with the parameters of the code, such as length and density. It would be interesting to investigate whether this observation can be confirmed by an analytic proof.

2) Behavior as a Function of SNR: Here we present simulations for adaptive LP decoding at different SNR levels. These experiments were performed for a randomly generated $(3,6)$-regular LDPC code of length 240, with its 4-cycles removed. At each SNR value, we simulated 2400 codeword transmissions.

In Figs. 5 and 6, the number of adaptive LP decoding iterations and the number of parity inequalities in the final iteration are plotted, respectively. The maximum and minimum values observed among the 2400 trials at each SNR are also included in the figures (solid lines marked with triangles). In addition, we have plotted 95% confidence intervals in these figures (dot-dashed lines). The boundaries for each of these confidence intervals are defined such that the right and left tails of the data histogram outside the interval each contain 2.5% of the population.

As we observe in Figs. 5 and 6, in the region where the SNR values are larger than the threshold of belief propagation (BP) decoding for the ensemble of $(3,6)$-regular LDPC codes, which is equal to 1.11 dB, the average numbers of iterations and constraints both decrease monotonically with SNR. This means that, similar to the BP decoder, the complexity of adaptive LP decoding decreases, on average, as the SNR increases beyond the threshold.

An interesting observation that we can make about both of these two figures is that the confidence intervals of the observed values are widest at an SNR in the range of 1 to 2 dB. To investigate this phenomenon, in Fig. 7, we have plotted the histogram of the number of parity inequalities in the last iteration of adaptive LP decoding at the SNR values of 0, 1.5, and 3 dB, which

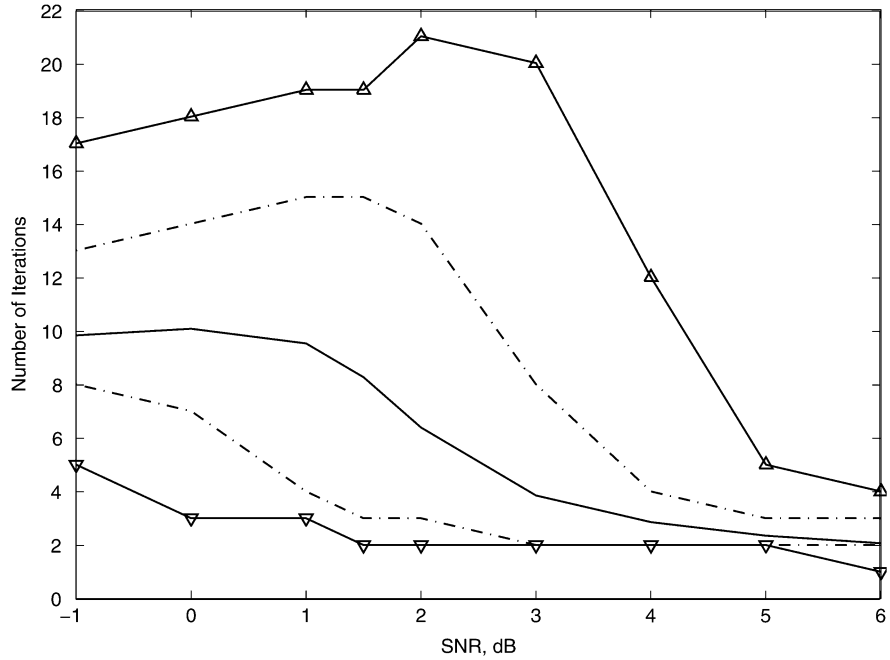

Fig. 5. The number of adaptive LP decoding iterations versus SNR for a $(3, 6)$-regular LDPC codes of length 240, with 2400 trials at each SNR. The solid line is the average number of iterations, the marked lines are the maximum and minimum values observed, and the dot-dashed lines indicate the 95% confidence interval.

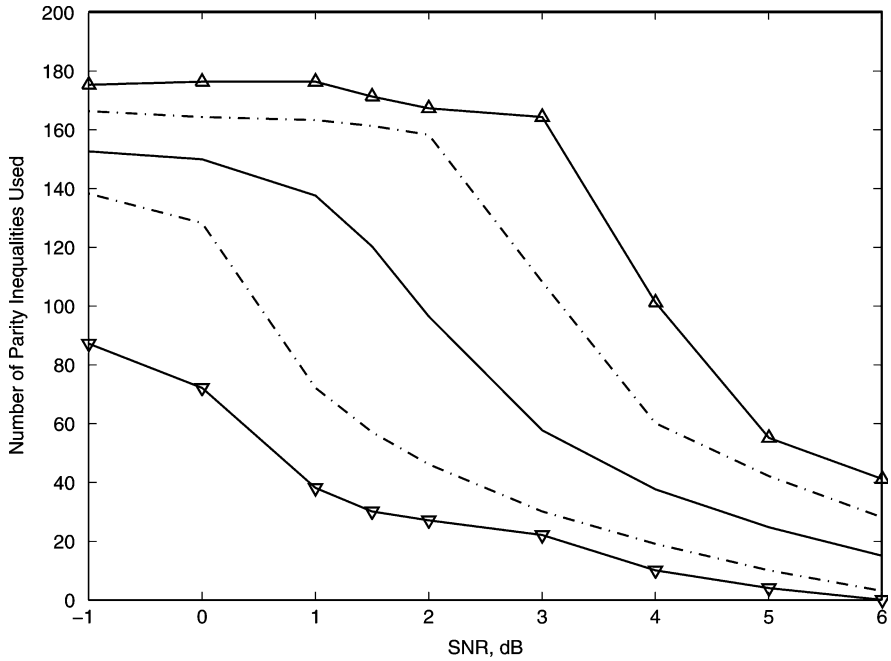

Fig. 6. The number of parity inequalities at the last iteration of adaptive LP decoding versus SNR for a $(3, 6)$-regular LDPC code of length 240, with 2400 trials at each SNR. The solid line is the average number of iterations, the marked lines are the maximum and minimum values observed, and the dot-dashed lines indicate the 95% confidence interval.

correspond to word error rate (WER) values of $0.966, 0.479$, and 0.0141, respectively. At SNR $= 0$ dB, where the decoder almost always outputs a fractional pseudocodeword, the values are concentrated around 150. However, at SNR $= 1.5$ dB, where the decoding succeeds about half the time, there are two distinct segments of roughly the same total mass in the histogram, which results in a wide confidence interval. Finally, at SNR $= 3$ dB, where the decoder is successful most of the time, the peak around 150 seen in the two previous histograms almost vanishes. These observations lead us to the natural conclusion that the complexity of adaptive LP decoding is much lower when it succeeds to find an integral (i.e., the ML) codeword.

IV. Generating Cuts to Improve the Performance

The complexity reduction obtained by adaptive LP decoding inspires further use of cutting-plane techniques to improve the error-rate performance of the algorithm. Specifically, when LP with all the original constraints gives a nonintegral solution, we

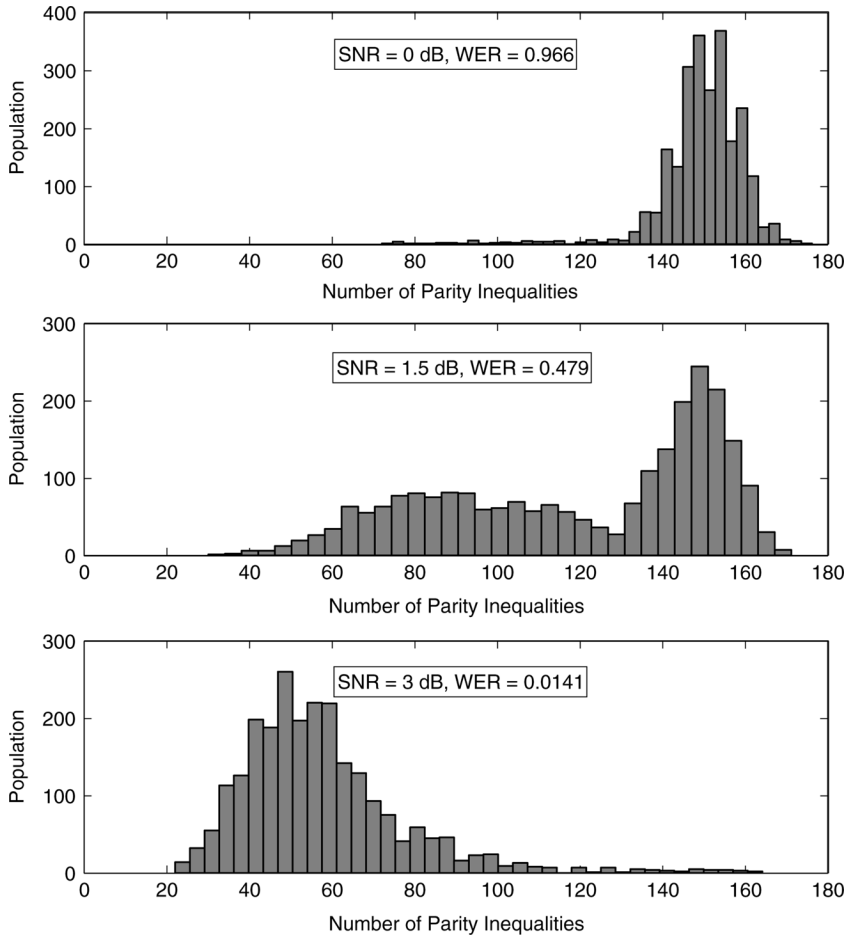

Fig. 7. The histogram of the number of parity inequalities at the last iteration of LP decoding at the SNR values of 0, 1.5, and 3.0 dB, each over a total population of 2400 samples.

try to cut the current solution, while keeping all the possible integral solutions (codewords) feasible.

The new cuts can be chosen from a pool of constraints describing a relaxation of the ML decoding problem which is tighter than the original relaxation defining the fundamental polytope. In this sense, this technique is similar in concept to the adaptive LP decoding presented in the previous section, with the difference that here there are more constraints to choose from. The effectiveness of this method depends on how closely the new relaxation approximates the ML decoding problem, and how efficiently we can search for those constraints that introduce cuts. Feldman *et al.* [1] have mentioned some ways to tighten the relaxation of the ML decoding problem, including the addition of parity inequalities associated to redundant parity checks (RPC) and the use of lift-and-project methods. (For more on lift-and-project techniques, see [15] and references therein.) Gomory's algorithm [16] is another well-known technique for solving general integer optimization problems, although it suffers from slow convergence. Each of these methods can be applied adaptively in the context of cutting-plane techniques. In this work, we focus on cutting-plane algorithms that use RPC cuts.

An RPC is obtained by modulo-2 addition of some of the rows of the parity-check matrix, and this new check introduces a number of constraints that may include a cut, which we call an *RPC cut*. Since each row of the parity-check matrix is a codeword from the dual code, adding an RPC is equivalent to including another dual codeword in the parity-check matrix. The simple structure of RPCs makes them an interesting choice for generating cuts. There are examples, such as the dual code of the $(7, 4)$ Hamming code, where even the relaxation obtained by adding all the possible RPC constraints (i.e., all dual codewords) does not guarantee convergence to a codeword. In other words, it is possible to obtain a nonintegral solution for which there is no RPC cut.

Kashyap [17] used the theory of matroids to show that for a certain class of codes, called *geometrically perfect* codes, the LP relaxation of ML decoding is exact if all the dual codewords are included in the parity-check matrix. Furthermore, although solving this relaxation by LP has exponential complexity, Kashyap showed that geometrically perfect codes can indeed be ML-decoded in polynomial time using a combinatorial algorithm. He also showed that, unfortunately, geometrically perfect codes are not asymptotically good. Hence, an important question that requires further study is how close we can get to the ML decoding of asymptotically good codes by adding all possible RPCs to the LP decoder. Also, finding efficient methods to search for RPC cuts for a given nonintegral solution remains an open problem. On the other hand, as observed in simulation results, RPC cuts are generally strong, and

finding a reasonable number of them often makes the resulting LP relaxation tight enough to converge to an integer-valued solution. In the following subsection, we propose and study some ideas for finding RPC cuts.

A. Finding Redundant Parity-Check Cuts

As mentioned before, there is an exponential number of RPCs that can be added, and in general, most of them do not introduce cuts. Hence, we need to find the cut-inducing RPCs efficiently by exploiting the special structure of the decoding problem. In particular, we will now demonstrate that cycles in the Tanner graph of the parity-check matrix have an important role in determining whether an RPC generates a cut. We start with some useful definitions.

Definition 2: Given a current nonintegral solution $\underline{x}$ of the relaxed LP problem, the subset $T \subseteq \{1, 2, \ldots, m\}$ of check nodes is called a *cut-generating collection* if the RPC defined by modulo-2 addition of the parity checks corresponding to T introduces a cut. If no proper subset of T other than itself has this property, we call it a *minimal cut-generating collection*.

Definition 3: Given a pseudocodeword $\underline{x}$, consider the set of variable nodes in the Tanner graph of the parity-check matrix whose corresponding elements in $\underline{x}$ have fractional values. Let F be the subgraph made up of these variable nodes, the check nodes directly connected to them, and all the edges that connect them. We call F the *fractional subgraph* and any cycle in F a *fractional cycle* at $\underline{x}$.

1) Role of Fractional Cycles: Theorem 3 below makes clear the relevance of the concept of fractional cycles to the search for cut-generating RPCs. Its proof makes use of the following lemma.

Lemma 2: Suppose that c_1 and c_2 are two parity checks whose corresponding parity inequalities are satisfied by the current solution $\underline{x}$. Then, $c \triangleq c_1 \oplus c_2$, the modulo-2 combination of these checks, can generate a cut only if the neighborhoods of c_1 and c_2 have at least two fractional-valued variable nodes in common.

Proof: See Appendix A. □

Theorem 3: Let $\underline{x}$ be a point satisfying all the parity inequalities induced by the check nodes in the Tanner graph, and let T be a collection of check nodes. If T is a cut-generating collection at $\underline{x}$, then there exists a fractional cycle such that all the check nodes on it belong to T.

Proof: We first consider the case where T is a minimal cut-generating collection. Note that any cut-generating collection must contain at least two check nodes, since no single check node generates a cut. Pick an arbitrary check node c_j in T. We make an RPC c_r by linearly combining the parity checks in $T \backslash \{c_j\}$. According to Lemma 2 and the minimality of T, there must be at least two fractional-valued variable nodes that are involved in both c_j and c_r. Since the set of variable nodes involved in c_r is a subset of the union of the neighborhoods of the check nodes in $T \backslash \{c_j\}$, if follows that c_j shares at least two fractional-valued neighbors with these check nodes. Let G be the subgraph of the Tanner graph consisting of the check nodes in T, their neighboring variable nodes, and the edges that connect them. Applying the above reasoning to every check node c_j in T, we conclude that each check node in the collection T is connected to at least two fractional-valued variable nodes of degree at least 2 within the subgraph G. Therefore, if we further remove all the integer-valued variable nodes, as well as all the fractional-valued variable nodes of degree less than 2 from G, we will be left with a number of check nodes and (fractional-valued) variable nodes that each have degree at least 2. A simple counting of edges will show that such a graph, which is a subset of G and the original Tanner graph, must contain a fractional cycle.

If T is not a minimal cut-generating collection, then it must contain a minimal cut-generating collection T_0. To see this, observe that there must be a check node in T whose removal leaves a cut-generating collection of check nodes. Iteration of this check node-removal process must terminate in a nonempty minimal cut-generating collection T_0 containing at least two check nodes. The subgraph G_0 corresponding to T_0 is contained in the subgraph G corresponding to T, so a discussion similar to that above proves that G_0, and therefore, G, contain fractional cycle. □

Remark 3: In a related result, Vontobel and Koetter showed in [3] that adding a redundant parity check to the code representation by combining a set of rows in the parity-check matrix H can possibly modify the fundamental polytope only if the Tanner graph corresponding to this set of rows contains a cycle. Here, Theorem 3 makes a stronger claim, in the sense that it requires the existence of a cycle over fractional-valued nodes, and not just any cycle. In another work, Chertkov and Chernyak in [11] studied the role that the cycles in the Tanner graph play in the performance of both BP and LP decoding, and proposed a heuristic method for improving LP decoding by characterizing the "most relevant" cycles. Their approach was based on modifying the likelihood values of the bits, rather than modifying the constraints as we propose here.

2) Existence of Fractional Cycles: In Theorem 4 below, we show that a fractional cycle always exists for any noninteger pseudocodeword.[5]

Lemma 3: At any point $\underline{x}$ in the fundamental polytope of LP decoding, no check node of the corresponding Tanner graph is connected to exactly one fractional variable node.

Proof: See Appendix B. □

We can represent the fractional subgraph F corresponding to the pseudocodeword $\underline{x}$ by a disjoint union of connected components (or briefly, components) $\{F_i\}, i = 1, \ldots, K$, for some $K \geq 1$.

Theorem 4: Let $\underline{x} \in [0, 1]^n$ be a fractional pseudocodeword of LP decoding for a code represented by Tanner graph G, and let F denote the fractional subgraph corresponding to $\underline{x}$. Then each component $F_i, i = 1, \ldots, K$, of F contains a cycle.

Proof: Since $\underline{x}$ is a vertex of the fundamental polytope, it should be the unique solution to the system of equations formed

[5]However, it should be emphasized that not every RPC constraint obtained from a fractional cycle generates a cut.

by turning the active parity inequalities and box constraints at $\underline{x}$ into equations. Consider a component F_i of the fractional subgraph, containing v (fractional) variable nodes. Clearly, none of the box constraints corresponding to these v variable nodes are active at $\underline{x}$. Hence, there should be v linearly independent active parity inequalities, introduced by the check nodes in F_i, that uniquely determine the values corresponding to the variable nodes in F_i in terms of the rest of the variables. Let us denote by $F_i' \subseteq F_i$ the subgraph obtained by removing all the check nodes in F_i which do not introduce any active parity inequality at $\underline{x}$, and all the edges connected to them.

A consequence of Lemma 1 is that, if a parity inequality κ given by a check node j is active at vertex $\underline{x}$, the rest of the parity inequalities introduced by check node j are not necessary to define vertex $\underline{x}$, since they cannot be violated at any point $\underline{x}_0 \in [0,1]^n$ where κ is active. Hence, to uniquely determine the values of the v variable nodes in F_i', it is sufficient to consider only one active constraint given by each check node in F_i'. We conclude that, in order to have enough equations, the number of check nodes in F_i', denoted by u, cannot be smaller than v. According to Lemma 3, each of these check nodes should be connected to at least two variable nodes in F_i'. Therefore, the number of edges in F_i' is at least $2u$, while the total number of nodes in this subgraph is $u+v \leq 2u$. This means that F_i' cannot be a tree or a forest, since it contains at least as many edges as nodes. Consequently, F_i', and hence, F_i, contain a cycle.

3) Randomized Algorithm for Finding RPC Cuts: The preceding results motivate the following algorithm to search for RPC cuts.

Algorithm 3:

Step 1: Given an LP decoding solution $\underline{x}$ and the original Tanner graph of the code, prune the graph by removing all the variable nodes with integer values.

Step 2: Starting from an arbitrary check node, randomly walk through the pruned graph until a cycle is found.

Step 3: Create an RPC by combining the rows of the parity-check matrix corresponding to the check nodes in the cycle.

Step 4: If this RPC introduces a cut, add this cut to the set of constraints in the LP relaxation, and exit; otherwise go to Step 2.

When the fractional subgraph contains many cycles and it is feasible to check only a small fraction of them, the randomized method described above can efficiently find cycles. However, when the cycles are few in number, this algorithm may actually check a number of cycles several times, while skipping some others. In this case, a structured search, such as one based on the depth-first search (DFS) technique, can be used to find all the simple cycles in the fractional subgraph. We can then check to see if any of them introduces an RPC cut. However, to guarantee that all the potential cut-generating RPCs are checked, one will still need to modify this search to include complex cycles, i.e., clusters of cycles which share some edges.

As shown above, by exploiting some of the properties of the linear code LP decoding problem, one can expedite the search for RPC cuts. However, there remains a need for more efficient methods of finding RPC cuts.

B. Complexity Considerations

There are a number of parameters that determine the complexity of the adaptive algorithm with RPC cuts, including the number of iterations of Algorithm 3 to find a cut, the total number of cuts that are needed to obtain an integer solution, and the time taken by each run of the LP solver after adding a cut. In particular, we have observed empirically that a number of cuts less than the length of the code is often enough to ensure convergence to the ML codeword. By using each solution of the LP as a warm start for the next iteration after adding further cuts, the time that each LP takes can be significantly reduced. For example, for a $(3,4)$-regular code of length 100 with RPC cuts, although as many as 70 LP problems may have to be solved for a successful decoding, the total time that is spent on these LP problems is no more than 10 times that of solving the standard problem (with no RPC cuts). Moreover, if we allow more than one cut to be added per iteration, the number of these iterations can be further reduced.

Since Algorithm 3 involves a random search, there is no guarantee that it will find a cut (if one exists) in a finite number of iterations. In particular, we have observed cases where, even after a large number of iterations, no cut was found, while a number of RPCs were visited repeatedly. This could mean that either no RPC cut exists for these cases, or the cuts have a structure that makes them unlikely to be selected by our random search algorithm.

In order to control the complexity, we can impose a limit, $C^{\max}$, on the number of iterations of the search, and if no cut is found after $C^{\max}$ trials, we declare failure. By changing $C^{\max}$, we can trade complexity for performance. Alternatively, we can put a limit, $T^{\max}$, on the total time that is spent on the decoding process. In order to find a proper value for this maximum, we ran the algorithm with a very large value of $C^{\max}$ and measured the total decoding time for the cases where the algorithm was successful in finding the ML solution. Based on these observations, we found that 10 times the worst case running time of the adaptive LP decoding algorithm of Section III serves as a suitable value for $T^{\max}$.

C. Numerical Results

To demonstrate the performance improvement achieved by using the RPC-based cutting-plane technique, we present simulation results for random $(3,4)$-regular LDPC codes on the AWGN channel. We consider codes of length 32, 100, and 240 bits, and in each case, we count about 200 failures to estimate the WER.

In Fig. 8, for the length-32 code, we plot the WER versus SNR for different values of $C^{\max}$, demonstrating the tradeoff between performance and complexity. As in the previous section, the SNR is defined as the ratio of the variance of the transmitted discrete-time signal to the variance of the noise sample.

For purposes of comparison, the WER of LP decoding with no RPC cut, as well as a lower bound on the WER of the ML

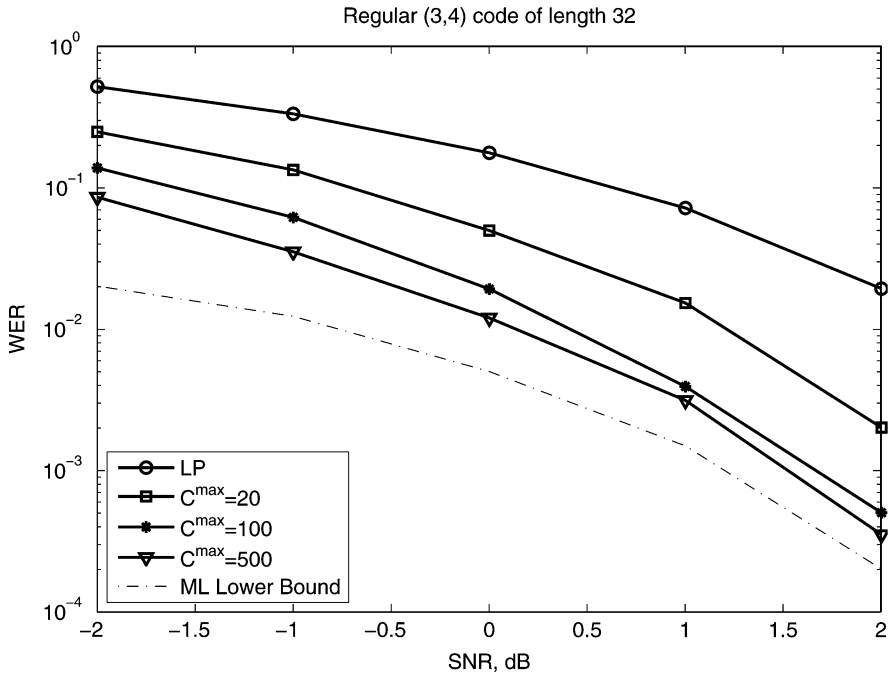

Fig. 8. WER of RPC-based cutting-plane LP versus SNR for different values of $C^{\max}$.

decoder, have been included in the figure. In order to obtain the ML lower bound, we counted the number of times that the RPC-based cutting-plane LP algorithm, using a large value of $C^{\max}$, converged to a codeword other than the transmitted codeword, and then divided that by the total number of transmitted codewords. Due to the ML certificate property of LP decoding, we know that ML decoding would fail in those cases, as well. On the other hand, ML decoding may also fail in some of the cases where LP decoding does not converge to an integral solution. Therefore, this estimate gives a lower bound on the WER of ML decoding.

However, for codes of length greater than 32, in almost all instances where the ML decoder had an error, an RPC-based LP decoder also failed to provide an integral (i.e., ML) solution. Therefore, it was not possible to obtain an estimate of the performance of the ML decoder using the above technique. Therefore, as an alternative, we used the performance of the Box-and-Match soft decision decoding algorithm (BMA) developed by Valembois and Fossorier [18] as an approximation of the ML decoder performance. For the codes that we consider, the error probability of BMA is guaranteed to be within a factor 1.05 of that of ML decoding.

In Figs. 9–11, the performance of LP decoding with RPC cuts is compared to that of standard LP decoding, sum–product decoding, and also the BMA (for the first two figures). Each figure corresponds to a fixed block length, and in all three cases sum–product decoding had 100 iterations. In these scenarios, instead of enforcing a limit on the iterations, we allow each decoding to take at most 10 times the average time required by the Adaptive LP decoder of Section III. The curves show that, as the SNR increases, the proposed method outperforms the LP and the SPA, and significantly closes the gap to the ML decoder performance. However, one can see that, as the code length increases, the relative improvement provided by RPC cuts becomes less pronounced. This may be due to the fact that, for larger code lengths, the Tanner graph becomes more tree-like, and therefore the negative effect of cycles on LP and message-passing decoding techniques becomes less important, especially at low SNR.

V. Conclusion

In this paper, we studied the potential for improving LP decoding, both in complexity and performance, by using an adaptive approach. Key to this approach is the ML certificate property of LP decoders, that is, the ability to detect the failure to find the ML codeword. This property is shared by message-passing decoding algorithms only in specific circumstances, such as on the erasure channel. The ML certificate property makes it possible to selectively and adaptively add only those constraints that are "useful," depending on the current status of the LP decoding process.

Using a cutting-plane approach, we proposed an adaptive LP decoding algorithm with decoding complexity that is independent of the code degree distributions, making it possible to apply LP decoding to parity-check codes of arbitrary densities. However, since general-purpose LP solvers are used at each iteration, the complexity is still a super-linear function of the block length, as opposed to a linear function as achieved by the message-passing decoder. It remains an open question whether special LP solvers for decoding of LDPC codes might take advantage of the sparsity of the constraints and other properties of the LP decoding problem to provide linear complexity.

In the second part of the paper, we explored the application of cutting-plane techniques for improving the error rate of LP decoding. We showed that redundant parity checks provide strong cuts, even though they may not guarantee ML performance. The

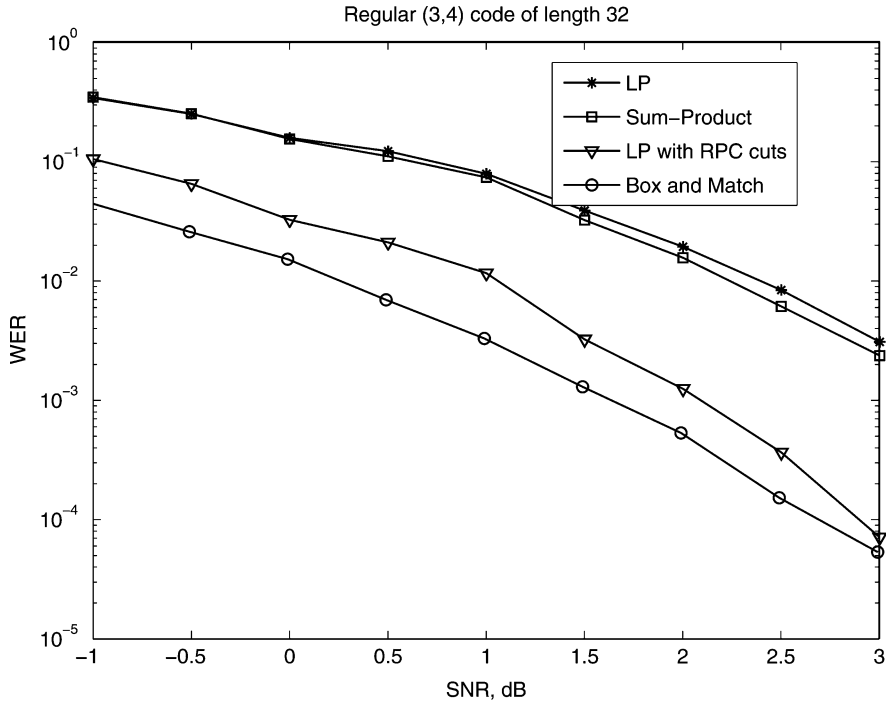

Fig. 9. WER of RPC-based cutting-plane LP versus SNR for length 32 and maximum decoding time 10 times that of LP decoding.

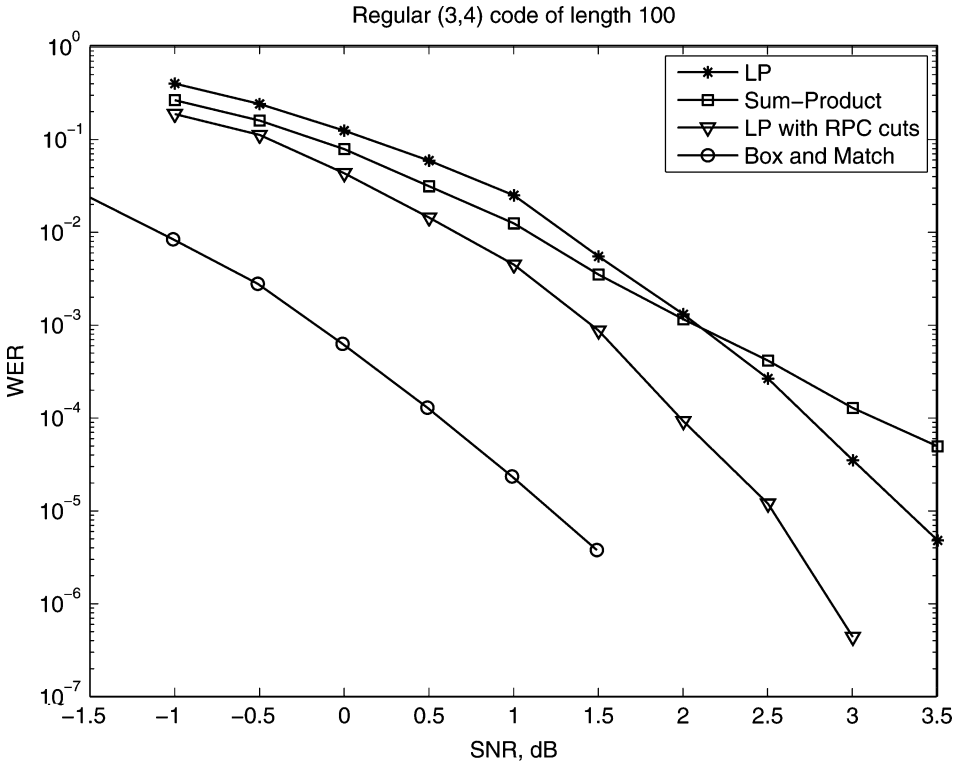

Fig. 10. WER of RPC-based cutting-plane LP versus SNR for length 100 and maximum decoding time 10 times that of the LP decoding.

results indicate that it would be worthwhile to find more efficient ways to search for strong RPC cuts by exploiting their properties, as well as to determine specific classes of asymptotically good codes for which RPC cuts are particularly effective. It would also be interesting to investigate the effectiveness of cuts generated by other techniques, such as lift-and-project cuts, Gomory cuts, or cuts specially designed for this decoding application.

APPENDIX A
PROOF OF LEMMA 2

Proof: Let N_1, N_2, and N denote the sets of variable nodes in the neighborhoods of c_1, c_2, and c, respectively, and define $S \triangleq N_1 \cap N_2$. Hence, we will have $N = (N_1 \cup N_2) \backslash S$. We can write S as a union of disjoint sets $S^{(0)}, S^{(1)}$, and $S^{(f)}$, which respectively, represent the members of S whose values in $\underline{x}$ are

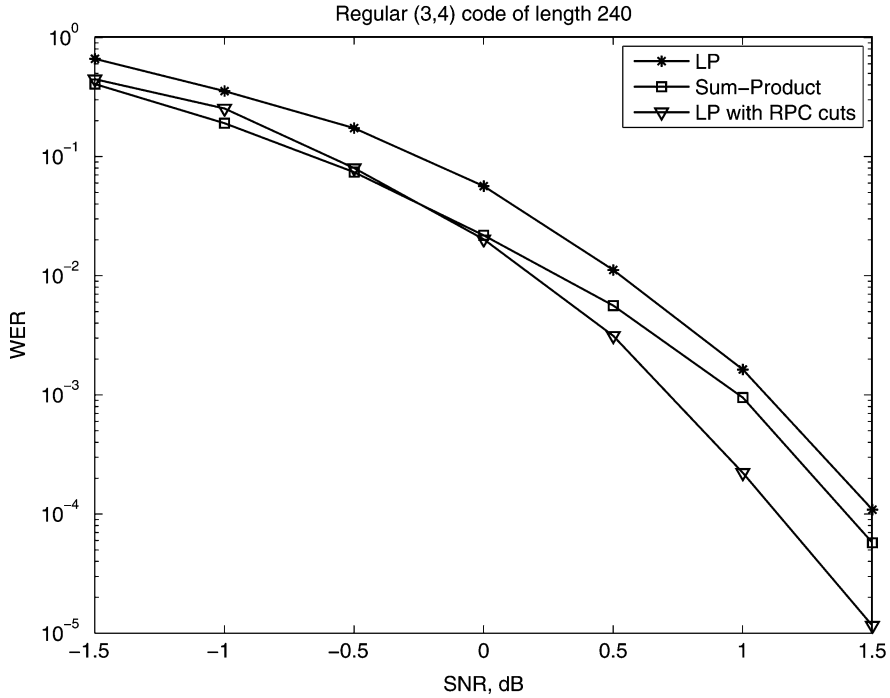

Fig. 11. WER of RPC-based cutting-plane LP versus SNR for length 240 and maximum decoding time 10 times that of the LP decoding.

equal to 0, equal to 1, or lie in the range $(0,1)$. In order to prove the lemma, it is enough to show that $|S^{(f)}| \geq 2$.

Parity check c generates a cut, hence there is an odd-sized subset $V \subseteq N$ of its neighborhood for which we can write

$$\sum_{i\in V} x_i - \sum_{i\in N\backslash V} x_i > |V| - 1. \tag{14}$$

Since N_1 and N_2 have no common member in N, we can write

$$V = V_1 \cup V_2 \tag{15}$$

where $V_1 \subseteq N_1\backslash S$ and $V_2 \subseteq N_2\backslash S$ are disjoint, and exactly one of them has odd size. Since c_1 and c_2 do not generate cuts, all the constraints they introduce should be satisfied by $\underline{x}$. Now consider the two sets $V_1 \cup S^{(1)}$ and $V_2 \cup S^{(1)}$. As $S^{(1)}$ is disjoint from both V_1 and V_2, and exactly one of V_1 and V_2 is odd-sized, we further conclude that exactly one of $V_1 \cup S^{(1)}$ and $V_2 \cup S^{(1)}$ is odd-sized, as well. Without loss of generality, assume that $|V_1 \cup S^{(1)}|$ is odd and $|V_2 \cup S^{(1)}|$ is even.

We proceed by dividing the problem into two cases, corresponding to whether $|S^{(f)}|$ is even or odd.

Case 1 ($|S^{(f)}|$ Is Even): Noting that $|V_1| + |S^{(1)}| + |S^{(f)}|$ is odd, consider the following parity inequality given by c_1:

$$\sum_{i\in V_1} x_i + \sum_{i\in S^{(1)}} x_i + \sum_{i\in S^{(f)}} x_i - \sum_{i\in N_1\backslash(V_1\cup S^{(1)}\cup S^{(f)})} x_i \leq |V_1| + |S^{(1)}| + |S^{(f)}| - 1. \tag{16}$$

Since $\sum_{i\in S^{(1)}} x_i = |S^{(1)}|$ and $\sum_{i\in S^{(0)}} x_i = 0$, we can simplify (16) as

$$\sum_{i\in V_1} x_i + \sum_{i\in S^{(f)}} x_i - \sum_{i\in N_1\backslash(V_1\cup S)} x_i \leq |V_1| + |S^{(f)}| - 1. \tag{17}$$

Now, starting from (14) and (15), and using the fact that $N_1\backslash(V_1 \cup S) \subseteq N\backslash V$, we have

$$\begin{aligned} |V_1| + |V_2| - 1 &< \sum_{i\in V_1} x_i + \sum_{i\in V_2} x_i - \sum_{i\in N\backslash V} x_i \\ &\leq \sum_{i\in V_2} x_i + \sum_{i\in V_1} x_i - \sum_{i\in N_1\backslash(V_1\cup S)} x_i \\ &\leq \sum_{i\in V_2} x_i + |V_1| - 1 + |S^{(f)}| - \sum_{i\in S^{(f)}} x_i \end{aligned} \tag{18}$$

where for the last inequality we used (17). Since $\sum_{i\in V_2} x_i \leq |V_2|$, (18) yields

$$|V_1| + |V_2| - 1 < |V_1| + |V_2| - 1 + |S^{(f)}| - \sum_{i\in S^{(f)}} x_i. \tag{19}$$

Hence, $S^{(f)}$ is a nonempty set, and since it is even in size, we must have $|S^{(f)}| \geq 2$.

Case 2 ($|S^{(f)}|$ Is Odd): We start by writing two parity inequalities induced by c_1 and c_2, which are, by assumption, satisfied at $\underline{x}$. For c_2 we write

$$\sum_{i\in V_2} x_i + \sum_{i\in S^{(1)}} x_i + \sum_{i\in S^{(f)}} x_i - \sum_{i\in N_2\backslash(V_2\cup S^{(1)}\cup S^{(f)})} x_i \leq |V_2| + |S^{(1)}| + |S^{(f)}| - 1 \tag{20}$$

and for c_1

$$\sum_{i\in V_1} x_i + \sum_{i\in S^{(1)}} x_i - \sum_{i\in N_1\backslash(V_1\cup S^{(1)}\cup S^{(f)})} x_i - \sum_{i\in S^{(f)}} x_i \leq |V_1| + |S^{(1)}| - 1. \tag{21}$$

By adding (20) and (21), we obtain after some cancellation

$$\sum_{i\in V_1} x_i + \sum_{i\in V_2} x_i - \sum_{i\in N_1\backslash(V_1\cup S)} x_i - \sum_{i\in N_2\backslash(V_2\cup S)} x_i \leq |V_1| + |V_2| + |S^{(f)}| - 2. \tag{22}$$

Note that

$$N\backslash V = [N_1\backslash(V_1 \cup S)] \cup [N_2\backslash(V_2 \cup S)]. \tag{23}$$

Now, (14) can be rewritten as

$$\sum_{i\in V_1} x_i + \sum_{i\in V_2} x_i - \sum_{i\in N_1\backslash(V_1\cup S)} x_i - \sum_{i\in N_2\backslash(V_2\cup S)} x_i > |V_1| + |V_2| - 1. \tag{24}$$

Combining (24) and (22) yields

$$|V_1| + |V_2| - 1 < |V_1| + |V_2| + |S^{(f)}| - 2 \tag{25}$$

and we conclude that $|S^{(f)}| > 1$. □

Appendix B
Proof of Lemma 3

By Contradiction: Assume, to the contrary, that at a point $\underline{x} \in \mathfrak{P}$, some check node $j^* \in \{1,\ldots,m\}$ is connected to exactly one fractional variable node, $i^* \in \{1,\ldots,n\}$, with the corresponding value $0 < x_{i^*} < 1$. We show that one of the parity inequalities in (3) introduced by the check node j^* must be violated at $\underline{x}$, contradicting the assumption that $\underline{x} \in \mathfrak{P}$.

Let A denote the set of integral neighbors of the check node j^* with value 1. We consider two cases for the cardinality of A.

Case 1 ($|A|$ Is Odd): Setting $V = A$, we find that

$$\sum_{i\in V} x_i - \sum_{i\in N\backslash V} x_i = V - x_{i^*} > |V| - 1. \tag{26}$$

Case 2 ($|A|$ Is Even): Setting $V = A \cup \{i^*\}$, we find in this case that

$$\sum_{i\in V} x_i - \sum_{i\in N\backslash V} x_i = V - 1 + x_{i^*} > |V| - 1. \tag{27}$$

Consequently, in both cases, the check node j^* introduces a violation of a parity inequality (3). □

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful suggestions. The authors also acknowledge Marc Fossorier for providing the numerical results for the Box-and-Match algorithm.

References

[1] 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.

[2] P. O. Vontobel and R. Koetter, "On the relationship between linear programming decoding and min-sum algorithm decoding," in *Proc. Int. Symp. Information Theory and Its Applications*, Parma, Italy, Oct. 2004, pp. 991–996.

[3] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," *IEEE Trans. Inf. Theory*, submitted for publication.

[4] J. S. Yedidia, W. T. Freeman, and Y. Weiss, Understanding Belief Propagation and Its Generalizations, Mitsubishi Electric Research Labs., Tech. Rep. TR2001-22, Jan. 2002.

[5] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "MAP estimation via agreement on trees: Message-passing and linear programming," *IEEE Trans. Inf. Theory*, vol. 51, no. 11, pp. 3697–3717, Nov. 2005.

[6] S. C. Draper, J. S. Yedidia, and Y. Wang, "ML decoding via mixed-integer adaptive linear programming," in *Proc. Int. Symp. Information Theory*, Nice, France, Jun. 2007, pp. 1656–1660.

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

[8] K. Yang, X. Wang, and J. Feldman, "Cascaded formulation of the fundamental polytope of general linear block codes," in *Proc. Int. Symp. Information Theory*, Nice, France, June 2007, pp. 1361–1365.

[9] P. O. Vontobel and R. Koetter, "On low-complexity linear-programming decoding," *Europ. Trans. Telecommun.*, vol. 5, pp. 509–517, Aug. 2007.

[10] A. Dimakis and M. Wainwright, "Guessing facets: Polytope structure and improved LP decoder," in *Proc. Int. Symp. Information Theory*, Seattle, WA, Jul. 2006, pp. 1369–1373.

[11] M. Chertkov and V. Chernyak, "Loop calculus helps to improve belief propagation and linear programming decoding of LDPC codes," in *Proc. Allerton Conf. Communications, Control, and Computing*, Monticello, IL, Sep. 2006, pp. 31–40.

[12] J. Nocedal and S. J. Wright, *Numerical Optimization.*. Berlin, Germany: Springer-Verlag, 2006.

[13] *GNU Linear Programming Kit*, [Online]. Available: http://www.gnu.org/software/glpk

[14] *MOSEK Optimization Software*, [Online]. Available: http://www.mosek.com

[15] M. Laurent, "A comparison of the Sherali-Adams, Lovász-Schrijver and Lasserre relaxations for 0–1 programming," *Math. of Oper. Res.*, vol. 28, no. 3, pp. 470–496, 2003.

[16] R. Gomory, An Algorithm for Integer Solutions to Linear Programs, Princeton IBM Mathematical Res. Rep., 1, Nov. 1958.

[17] N. Kashyap, "A decomposition theory for binary linear codes," *IEEE Trans. Inf. Theory.*, vol. 54, no. 7, pp. 3035–3058, Jul. 2008.

[18] 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.