Efficient Implementation of Linear Programming Decoding

Mohammad H. Taghavi*, Amin Shokrollahi†, and Paul H. Siegel*

* University of California, San Diego, Email: (mtaghavi, psiegel)@ucsd.edu
† Ecole Polytechnique Fédérale de Lausanne (EPFL), Email: amin.shokrollahi@epfl.ch

***Abstract*—This paper explores ideas for fast linear programming (LP) decoding of low-density parity-check (LDPC) codes. We first propose a modification of Adaptive LP decoding, and prove that it performs LP decoding by solving a number of linear programs that contain at most one linear constraint derived from each of the parity-check constraints. Then, as a step toward designing an efficient LP solver that exploits the structure of LDPC codes, we study a sparse interior-point implementation for solving this sequence of linear programs. Since the most complex part of each iteration of the interior-point algorithms is to solve a (usually ill-conditioned) system of linear equations for finding the step direction, we propose a framework for designing preconditioners to be used with the iterative methods for solving these systems. The effectiveness of the proposed approach is demonstrated via both analytical and simulation results.**

I. Introduction

Linear programming (LP) decoding was proposed by Feldman et al. [1] as an alternative to iterative message-passing (IMP) decoding of LDPC and turbo-like codes. LP decoding approximates the maximum-likelihood (ML) decoding problem by a linear optimization problem via relaxing each of the finite-field parity-check constraints of the ML decoding into a number of linear constraints. Due to its geometric structure, LP decoding seems to be more amenable than IMP decoding to finite-length analysis. In particular, the finite-length behavior of LP decoding can be completely characterized in terms of *pseudocodewords*. Another characteristic of LP decoding – the *ML certificate property* – is that its failure to find an ML codeword is always detectable.

On the other hand, the main disadvantage of LP decoding is its higher complexity compared to IMP decoding. A conventional implementation of LP decoding is highly complex due to two main factors: (1) the large size of the LP problem formed by relaxation, and (2) the inability of general-purpose LP solvers to solve the LP efficiently by taking advantage of the properties of the decoding problem.

The standard formulation of LP decoding [1] has a size that grows very rapidly with the density of the parity-check matrix. Adaptive LP (ALP) decoding was proposed in [2] to address this problem, reducing LP decoding to solving a sequence of much smaller LP problems. The size of these LP problems has been observed in practice to be independent of the degree distribution, although this observation has not been analytically explained. More recently, an equivalent formulation of the LP decoding problem was proposed in [3] and [4], with a problem size growing linearly with both the code length and the maximum check node degrees.

In this paper, we take some steps toward designing efficient LP solvers for LP decoding that exploit the inherent sparsity and structure of this particular class of problems. Our approach is based on a sparse implementation of interior-point (IP) algorithms. In an independent work, Vontobel studied the implementation and convergence of IP methods for LP decoding and mentioned a number of potential approaches to reduce its complexity [5]. 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; e.g. see [6] and [7].

We first propose a modified version of ALP decoding, in which we adaptively remove a number of constraints at each iteration of ALP decoding, while adding new constraints to the problem. We prove a number of properties of this algorithm, which facilitate the design of a low-complexity LP solver. In particular, we show that the modified ALP decoder has the *single-constraint property*, which means that it performs LP decoding by solving a series of linear programs that each contain at most one linear constraint from each parity check.

Then, we focus on the most complex part of each iteration of the IP algorithm, which is solving a system of linear equations to compute the Newton step. Since these linear systems become ill-conditioned as the IP algorithm approaches the solution, iterative methods, such as the conjugate-gradient (CG) method, that are often used for solving sparse systems perform poorly in the later iterations of the optimization. As a solution, we propose a criterion for designing preconditioners that take advantage of the properties of LP decoding, along with a number of greedy algorithms to search for such preconditioners. The proposed preconditioning algorithms have similarities to the encoding procedure of LDPC codes, and we demonstrate their effectiveness via both analytical methods and computer simulation results.

The rest of this paper is organized as follows. In Section II, we review LP decoding and ALP decoding. In Section III, we propose the modified ALP decoding, and show a number of properties of this algorithm. In Section IV, we focus on the sparse implementation of an IP linear programming method, and the need for preconditioning to compute the Newton step in this algorithm. In Section V, we introduce a preconditioning approach to speed up the iterative computation

of the Newton step. Some theoretical analysis and computer simulation results are presented in Section VI. Section VII concludes the paper.

II. Linear Programming Decoding

A. LP Relaxation of Maximum-Likelihood Decoding

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

$$\textbf{ML Decoding} \quad \begin{array}{ll} \text{minimize} & \gamma^T u \\ \text{subject to} & u \in \mathcal{C}, \end{array} \tag{1}$$

where γ is the vector of log-likelihood ratios (LLR) given by

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

This is an optimization with a linear objective function, but with nonlinear constraints in $\mathbb{R}$. It is desirable to replace these constraints by a number of linear constraints, so that decoding can be performed using linear programming. The feasible space of the desired LP would be the convex hull of all the codewords in $\mathcal{C}$. Unfortunately, the number of constraints needed for this LP representation grows exponentially with the code length, making this approach impractical.

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 is known as the *fundamental polytope*, $\mathcal{P}$ [8]. To describe the (projected) fundamental polytope, linear constraints are derived from a parity-check matrix as follows. For each row $j = 1, \ldots, m$ of the parity-check matrix, i.e., each check node, the LP formulation includes the constraints

$$\sum_{i\in\mathcal{V}}(1-u_i)+\sum_{i\in\mathcal{N}(j)\setminus\mathcal{V}} u_i \geq 1, \ \ \forall\, \mathcal{V}\subseteq\mathcal{N}(j) \text{ s. t. } |\mathcal{V}| \text{ is odd.} \tag{3}$$

We refer to the constraints of this form as *parity inequalities*. If the variables u_i are zeroes and ones, these constraints will be equivalent to the original binary parity-check constraints. Now, given this equivalence, we relax the LP problem by replacing each binary constraint, $u_i \in \{0, 1\}$, by a *box constraint* of the form $0 \leq u_i \leq 1$. LP decoding can then be written as

$$\textbf{LP Decoding} \quad \begin{array}{ll} \text{minimize} & \gamma^T u \\ \text{subject to} & u \in \mathcal{P}. \end{array} \tag{4}$$

The integral vertices of the fundamental polytope exactly correspond to the codewords of $\mathcal{C}$. 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. On the other hand, if LP decoding gives as the solution one of the nonintegral vertices, which are called the *pseudocodewords*, the decoder declares a failure.

B. Adaptive Linear Programming Decoding

In [2] the adaptive LP (ALP) decoding algorithm was proposed as an alternative to the direct implementation of LP decoding (4). In this method, a hierarchy of LPs with the same objective function as in (4) are solved, with the solution to the last program being identical to that of LP decoding. The first LP in this hierarchy is made up of only n box constraints, such that for each $i \in \{1, 2, \ldots, n\}$, we include the constraint

$$\begin{cases} 0 \leq u_i & \text{if } \ \gamma_i \geq 0, \\ u_i \leq 1 & \text{if } \ \gamma_i < 0. \end{cases} \tag{5}$$

The solution to this initial problem corresponds to the result of (uncoded) bit-wise hard decisions on the received vector. The ALP decoding algorithm is written in Algorithm 1.

Algorithm 1 ALP Decoding

1: Setup the initial LP problem with constraints from (5), and $k \leftarrow 0$;
2: Find the solution u^0 to the initial LP problem by bit-wise hard decision;
3: **repeat**
4: $\quad k \leftarrow k + 1$;
5: $\quad$ Find the set $\mathcal{S}^k$ of all parity inequalities and box constraints that are violated at u^{k-1};
6: $\quad$ If $|\mathcal{S}^k| > 0$, add the constraints in $\mathcal{S}^k$ to the LP problem and solve it to obtain u^k;
7: **until** $|\mathcal{S}^k| = 0$
8: Output $u = u^k$ as the solution to LP decoding.

In Step 5 of this algorithm, the search for all the violated parity inequalities can be performed efficiently using Algorithm 1 of [2], without having to examine all the $O(m2^{\,d_{max}})$ parity inequalities of the original LP decoding formulation.

In [2], the number of iterations of ALP decoding was upper-bounded by the code length, n. However, it was observed in the simulations that the typical number of iterations is much smaller in practice (less than 20 for all $n < 2000$). Moreover, one can conclude from the following theorem that, at each iteration of ALP decoding, the number of violated parity inequalities added to the problem is at most m.

Theorem 1 ([9]): If at any given point $u \in [0, 1]^n$, 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.

III. Modified ALP Decoding

Definition 1: A linear inequality constraint of the form $a^T x \leq b$ is called active at point x^0 if it holds with equality; i.e., $a^T x^0 = b$, and is called inactive if it holds with strict inequality; i.e. $a^T x^0 < b$.

The following is a corollary of Theorem 1

Corollary 1: If one of the parity inequalities introduced by a check node is active at a point $x^0 \in [0, 1]^n$, all parity inequalities from this check node must be satisfied at x^0.

Now consider the linear program LP^k at an iteration k of ALP decoding, with an optimum point u^k. It is easy to see

that among the constraints in this linear program, the inactive ones are *non-binding*, meaning that, if we remove the inactive constraints from the problem, u^k remains an optimum point of the feasible space. Motivated by the fact above, we propose the modified ALP decoding algorithm (MALP decoding), stated in Algorithm 2, where, after solving each LP, a subset of the constraints that are active at the solution are removed.

It was conjectured in [9] that no box constraint can be violated at any intermediate solution of ALP decoding. We will prove this conjecture for both ALP and MALP decoding in a forthcoming paper [10]. Hence, in this work, we do not search for violated box constraints in the proposed algorithm.

Algorithm 2 MALP Decoding

1: Setup the initial LP problem with constraints from (5), and $k \leftarrow 0$;
2: Find the solution u^0 to the initial LP problem by bit-wise hard decision;
3: **repeat**
4: $k \leftarrow k+1$; $flag \leftarrow 0$;
5: **for** $j = 1$ to m **do**
6: **if** check node j introduces a parity inequality that is violated at u^{k-1} **then**
7: Remove the parity inequalities of check node j (if any) from the current problem;
8: Add the new (violated) constraint to the LP problem; $flag \leftarrow 1$;
9: **end if**
10: **end for**
11: If $flag = 1$, solve the LP problem to obtain u^k;
12: **until** $flag = 0$
13: Output $u = u^k$ as the solution to LP decoding.

The LP problems solved in the ALP and MALP decoding algorithms can be written in the "standard" matrix form as

$$\begin{aligned} \text{minimize} \quad & \gamma^T u \\ \text{subject to} \quad & Au \leq b, \\ & u_i \geq 0 \quad \forall i \in \mathcal{I}: \ \gamma_i \geq 0, \\ & u_i \leq 1 \quad \forall i \in \mathcal{I}: \ \gamma_i < 0, \end{aligned} \tag{6}$$

where matrix A is called the *constraint matrix*.

A. The Single-Constraint Property

In Theorem 2 of [2], it has been shown that the sequence of solutions to the intermediate LP problems in ALP decoding converges to that of LP decoding in at most n iterations. Using a similar approach it can be easily shown that this property indeed holds for MALP decoding, as well.

Theorem 2: In the LP problem at any iteration k of the MALP decoding algorithm, there is at most one parity inequality corresponding to each check node of the Tanner graph.

By induction: The initial LP problem consists only of box constraints. So, it suffices to show that, if the LP problem LP^k at an iteration k satisfies the desired property, the LP problem LP^{k+1} in the subsequent iteration satisfies this property, as well. Consider check node j which has a violated parity inequality κ_j at the solution u^k of LP^k. According to Corollary 1, if there already has been a parity inequality $\tilde{\kappa}_j$ from this check node in LP^k, $\tilde{\kappa}_j$ cannot be active at u^k, hence, the MALP decoder will remove $\tilde{\kappa}_j$ before adding κ_j to LP^{k+1}. As a result, there cannot be more than one parity inequality from any check node j in LP^{k+1}. ■

Corollary 2: The number of parity inequalities in any linear program solved by the MALP decoder is at most m.

The result above is in contrast to the non-adaptive formulations of LP decoding, where the size of the LP problems grows with the check node degree. Hence, the complexity of MALP decoding can be bounded by its number of iterations times the worst-case complexity of solving an LP problem with n variables and m parity inequalities.

An important consequence of Theorem 2 is that, in the LP problems that are solved in MALP decoding, the distribution of the nonzero elements of the LP constraint matrix, A, has the same structure as that of the parity-check matrix, H, after removing the rows of H that are not represented by a parity inequality in the LP. This is due to the fact that the support set of a row of A, corresponding to a parity inequality, is identical to that of the row of H from which it has been derived, and in addition, each row of A is derived from a unique row of H. As we will see later in this paper, this property can be exploited in the design of efficient LP solvers.

The following corollary results from Corollary 2:

Corollary 3: The solution to any LP decoding problem differs in at most $n-m$ coordinates from the vector obtained by making bit-based hard decisions on the LLR vector, γ.

Proof: Omitted. ■

IV. Interior Point Method for Solving the LPs

In this and the next section, we investigate how an interior-point (IP) linear optimization algorithm can be implemented efficiently for LP decoding.

A. Implementation Issues of the Interior-Point Algorithms

For simplicity, in this section we assume that the LP problems we want to solve are of the form (6). However, by introducing additional slack variables, we can modify the expressions in a straighforward way to represent the case where both types of box constraints may be present for each variable.

We first write the LP problem with q variables and p constraints in the "augmented" form

$$\textbf{Primal LP} \qquad \begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0. \end{aligned} \tag{7}$$

Here, to convert the LP problem (6) into the form above, we have taken two steps. First, noting that each variable u_i in (6) is subject to exactly one box constraint of the form $u_i \geq 0$ or $u_i \leq 1$, we introduce the variable vector x and

cost vector c, such that for any $i = 1, \ldots, n$, $x_i = u_i$ and $c_i = \gamma_i$ if the former inequality is included (i.e., $\gamma_i \geq 0$), and $x_i = 1 - u_i$ and $c_i = -\gamma_i$, otherwise. Therefore, the box constraints will all have the form $x_i \geq 0$, and the coefficients of the parity inequalities will also change correspondingly. Second, for any $j = 1, \ldots, p$, we convert the parity inequality $A_{j\diamond}x \leq b_j$ in (6), where $A_{j\diamond}$ denotes the jth row of A, to a linear equation $A_{j\diamond}x + x_{n+j} = b_j$, by introducing p nonnegative slack variables $x_{n+1}, \ldots, x_q$, where $q = n + p$, with corresponding coefficients equal to zero in the cost vector, c. We will sometimes refer to the first n (non-slack) variables as the *standard variables*. The dual of the LP has the form

$$\textbf{Dual LP} \quad \begin{array}{ll} \text{minimize} & b^T y \\ \text{subject to} & A^T y + z = c, \\ & z \geq 0, \end{array} \tag{8}$$

where y and z are the dual standard and slack variables, respectively.

IP algorithms consist of a variety of algorithms, differing in the way the problem is approximated and how the step is calculated at each iteration. One of the most successful classes of IP methods is the primal-dual path-following algorithm, which is most effective for large-scale applications. It has been shown that the path-following algorithm converges to the solution in $O(\sqrt{n})$ iterations in the worst-case, while the number of iterations is typically $O(\log n)$ [11]. Therefore, if the Newton step at each iteration can be computed efficiently by taking advantage of the sparsity and structure in the problem, one could obtain an algorithm that is faster than the simplex algorithm for large-scale problems. In this paper, we will skip a review of the IP algorithm, and will only focus on its complexity bottleneck. For a comprehensive description, we refer the reader to the literature on linear programming and IP methods, e.g. [11].

Let $s = (x, y, z)$ be the current "iterate" (or state) of the IP algorithm solving the primal-dual LP problem given in (7), (8), and let X and S denote diagonal matrices, with the entries of x and z on their diagonal, respectively. The most complex part of the IP algorithm at each iteration is to solve a "normal" system of linear equations of the form

$$(AD^2A^T)\Delta_y = w \tag{9}$$

for Δ_y, where

$$D^2 = XZ^{-1}, \tag{10}$$

as part of the process of computing the direction for the next Newton step.

The primal-dual path-following algorithm will iterate until the duality gap $g_d \triangleq x^T z$ becomes smaller than some small constant $\epsilon > 0$. It has been shown that with a proper choice of the step lengths, this algorithm takes $O\left(\sqrt{q}\log(\epsilon_0/\epsilon)\right)$ steps to reduce the duality gap from ϵ_0 to ϵ.

In order to initialize the algorithm, we need some feasible $x^0 > 0$, y^0, and $z^0 > 0$. Obtaining such an initial point is nontrivial, and is usually done by introducing a few dummy variables, as well as a few rows and columns to the constraint matrix. This may not be desirable for a sparse LP, since the new rows and columns will not generally be sparse. Furthermore, if the Newton directions are computed based on the feasibility assumption, round-off errors can cause instabilities due to the gradual loss of feasibility. As an alternative, an infeasible variation of the primal-dual path-following algorithm is often used, where any $x^0 > 0$, y^0, and $z^0 > 0$ can be used for initialization. This algorithm will simultaneously try to reduce the duality gap and move the iterates into the feasible space.

B. Solving the Linear System: Preconditioning

The most complex step at each iteration of the IP algorithm in the previous subsection is to solve the normal system of linear equations in (9). While these equations were derived for the primal-dual path-following algorithm, in most other variations of IP methods, we encounter linear systems of similar form.

Suppose we want to find the solution x^* to an arbitrary system of linear equations given by

$$Qx = w, \tag{11}$$

where Q is a $q \times q$ symmetric positive definite matrix, which is equal to AD^2A^T in the case of (9). Various algorithms for solving such a system fall into two main categories of *direct methods* and *iterative methods*. While direct methods, such as Gaussian elimination, attempt to solve the system in a finite number of steps, and are exact in the absence of rounding errors, iterative methods start from an initial guess, and derive a sequence of approximate solutions. Since the constraint matrix Q is symmetric and positive definite, the most common direct method for solving this problem is based on computing the Cholesky decomposition of this matrix. However, this approach is inefficient for large-scale sparse problems, due to the computational cost of the decomposition, as well as loss of sparsity. Hence, in many sparse LP problems, e. g. network flow linear programs, iterative methods such as the conjugate gradient (CG) method [12] are preferred.

While in principle the CG algorithm requires q steps to find the exact solution x^*, sometimes a much smaller number of iterations provides a sufficiently accurate approximation to the solution. The distribution of the eigenvalues of the coefficient matrix Q has a crucial effect on the convergence behavior of the CG method. In particular, the number of iterations to reduce the residual error by a certain factor from its initial value can be upper-bounded by a constant times $\sqrt{\kappa(Q)}$ [13, Chapter 6], where $\kappa(Q)$ is the spectral condition number of Q, i.e. the ratio of the maximum and minimum eigenvalues of Q. However, the condition number is not the only factor determining the behavior of this algorithm. We henceforth call the matrix Q ill-conditioned, in loose terms, if the CG method converges slowly in solving (11).

In the IP algorithm, the spectral behavior of $Q = AD^2A^T$ changes as a function of the diagonal elements $d_1, \ldots, d_q$, of D, which are, as described in the previous subsection, the

square roots of the ratios between the primal variables $\{x_i\}$ and the dual slack variables $\{z_i\}$. In practice, at each iteration of the path-following IP method, the product $x_i z_i$ is relatively constant over all i, such that we have

$$d_i \approx \frac{1}{\sqrt{\mu}} x_i, \ \forall i = 1, \ldots, q, \tag{12}$$

where

$$\mu = \frac{x^T z}{q}. \tag{13}$$

is proportional to the duality gap. As the iterates of the IP algorithm become closer to the solution and μ approaches zero, many of the d_i's take very small or very large values, depending on the value of the corresponding x_i in the solution. This has a negative effect on the spectral behavior of Q, and as a result, the convergence of the CG method.

When the coefficient matrix Q is ill-conditioned, it is common to use preconditioning. In this method, we use a symmetric positive-definite matrix M as an approximation of Q, and instead of (11), we solve the equivalent preconditioned system

$$M^{-1}Qx = M^{-1}w. \tag{14}$$

A good preconditioner M needs to satisfy two requirements. First, $M^{-1}Q$ should have a better spectral distribution than Q, so that the preconditioned system can be solved faster than the original system. Second, it should be inexpensive to solve $Mx = z$, since we need to solve a system of this form at each step of the preconditioned algorithm. Therefore, a natural approach is to design a preconditioner which, in addition to providing a good approximation of Q, has an underlying structure that makes it possible to solve $Mx = z$ using a direct method in linear time.

V. Preconditioner Design for LP Decoding

Our approach for designing an effective preconditioner for LP decoding is to find a *preconditioning set*, $\mathcal{M} \subseteq \{1, \ldots, q\}$, corresponding to p columns of A and D, resulting in $p \times p$ matrices $A_{\mathcal{M}}$ and $D_{\mathcal{M}}$, such that $M = A_{\mathcal{M}} D_{\mathcal{M}}^2 A_{\mathcal{M}}^T$ is both easily invertible and a good approximation of $Q = AD^2A^T$.

In the product AD^2A^T, the ith column of A is scaled by the ith diagonal element d_i of the diagonal matrix D. Hence, to design an effective preconditioner, it is natural to select $\mathcal{M}$ to include the columns with the highest scale factors, or *weights*, $\{d_i\}$, while keeping $A_{\mathcal{M}}$ and $A_{\mathcal{M}}^T$ full rank and invertible in $O(q)$ time. Then, the solution x to $Mx = z$ can be found by sequentially solving $A_{\mathcal{M}} f_1 = z$, $D_{\mathcal{M}}^2 f_2 = f_1$, and $A_{\mathcal{M}}^T x = f_2$, for f_1, f_2, and x, respectively.

A. Preconditioning via Triangulation

For a sparse constraint matrix, A, a sufficient condition for $A_{\mathcal{M}}$ and $A_{\mathcal{M}}^T$ to be invertible in $O(q)$ time is that $A_{\mathcal{M}}$ can be made upper or lower triangular, with nonzero diagonal elements, using column and/or row permutations. We call a preconditioning set $\mathcal{M}$ that satisfies this property a *triangular set*. Once an upper- (lower-) triangular form $A_{\mathcal{M}}^{\triangle}$ of $A_{\mathcal{M}}$ is found, we start from the last (first) row of $A_{\mathcal{M}}^{\triangle}$, and, by taking advantage of the sparsity, solve for the variable corresponding to the diagonal element of each row recursively in $O(1)$ time. It is not difficult to see that there always exists at least one triangular set for any LP decoding problem; one example is the set of columns corresponding to the slack variables, which results in a diagonal $A_{\mathcal{M}}$.

As a criterion for finding the best approximation $A_{\mathcal{M}} D_{\mathcal{M}}^2 A_{\mathcal{M}}^T$ of AD^2A^T, we search for the maximum-weight triangular set (MTS), i.e. the triangular set that contains the columns with the highest weights, d_i. We propose a greedy approach, to search for the MTS, motivated by the properties of the LP decoding problem.

The problem of bringing a parity-check matrix into (approximate) triangular form has been studied by Richardson and Urbanke [14] in the context of the encoding of LDPC codes. The authors proposed a series of greedy algorithms that are similar in nature to the peeling algorithm for decoding in the binary erasure channel: repeatedly select a nonzero entry (edge) of the matrix (graph) lying on a degree-1 column or row (variable or check node), and remove both the column and row of this entry from the matrix. They showed that parity-check matrices that are optimized for erasure decoding can be made almost triangular using this greedy approach.

The fact that the constraint matrices of the LP problems in MALP decoding have structure similar to the corresponding parity-check matrix motivates the use of a greedy algorithm analogous to those in [14] for triangulating the matrix A. However, this problem is different from the encoding problem, in that we are not merely interested in making A triangular, but rather, we look for the triangular submatrix with the maximum weight. In fact, as mentioned earlier, finding one triangular form of A is trivial, due to the presence of the slack variables. Here, we present two greedy algorithms to search for the MTS, one of which is related to the algorithms of Richardson and Urbanke. Throughout this section, we will also refer to the outputs of these (suboptimal) greedy algorithms, in loose terms, as the MTS, although they may not necessarily have the maximum possible weight.

1) Incremental Greedy Search for the MTS: Although an ideal preconditioning set would contain the q columns of the matrix that have the q highest weights, in reality, the square submatrix of A comprised of these q columns is often neither triangular nor full rank. In the incremental greedy search for the MTS, we start by selecting the highest-weight column, and try to expand the set of selected columns by giving priority to the columns of higher weights, while maintaing the property that the corresponding submatrix can be made lower-triangular by column and row permutations.

Let $\mathcal{S}$ be a set of selected columns from A, where $|\mathcal{S}| \leq p$. In order to check whether the submatrix $A_{\mathcal{S}}$ can be made lower-triangular by column and row permutations, we can treat the variable nodes corresponding to $\mathcal{S}$ in the Tanner graph as erased bits, and use the peeling algorithm to decode them in $O(q)$ time. We call this process the Triangulation Step, and assume that, if full triangulation is possible, this procedure outputs an $|\mathcal{S}| \times |\mathcal{S}|$ lower-triangular submatrix $A_{\mathcal{S}}^{\triangle}$.

Using the Triangulation Step as a subroutine, the incremental greedy search method, given by Algorithm 3, first sorts the columns according to their corresponding weights, d_i (or, alternatively, x_i), and initializes the preconditioning set, $\mathcal{M}$, as an empty set. Starting with the highest-weight column and going down the sorted list of column indices, it adds each column to $\mathcal{M}$ if the submatrix corresponding to the resulting set can be made lower triangular using the triangulation step.

Algorithm 3 Incremental Greedy Search for the MTS

1: **Input:** $p \times q$ constraint matrix A, and the set of column weights, $d_1 \dots d_q$;
2: **Output:** A triangular set $\mathcal{M}$ and the $p \times p$ lower-triangular matrix $A^{\triangle}_{\mathcal{M}}$;
3: **Initialization:** $\mathcal{M} \leftarrow \emptyset$, $i \leftarrow 0$;
4: Sort the column indices $\{1, \dots, q\}$ according to their corresponding weights, d_i, in decreasing order, to obtain the permuted sequence $\pi_1, \dots, \pi_q$, such that $d_{\pi_1} \geq \dots \geq d_{\pi_q}$;
5: **while** $i < q$ and $|\mathcal{M}| < p$ **do**
6: $\quad i \leftarrow i + 1$, $\leftarrow \mathcal{M} \cup \{\pi_i\}$;
7: $\quad$ **if** the Triangulation Step can bring the submatrix $A_{\mathcal{S}}$ into the lower-triangular form $A^{\triangle}_{\mathcal{S}}$ **then**
8: $\quad\quad \mathcal{M} \leftarrow \mathcal{S}$, $A^{\triangle}_{\mathcal{M}} \leftarrow A^{\triangle}_{\mathcal{S}}$;
9: **end while**

We claim that, due to the presence of the slack columns in A, Algorithm 3 will successfully find a triangular set $\mathcal{M}$ of p columns; i.e., it exits the while-loop (lines 5-9) only when $|\mathcal{M}| = p$. Assume, on the contrary, that the algorithm ends while $|\mathcal{M}| < p$, so that the matrix $A_{\mathcal{M}}$ is a $p \times |\mathcal{M}|$ lower-triangular matrix. This means that if we add any column $k \in \{1, \dots, q\} \backslash \mathcal{M}$ to $\mathcal{M}$, it cannot be made lower triangular, since otherwise, column k would have already been added to $|\mathcal{M}|$ when $\pi_i = k$ in the while-loop.[1] However, this clearly cannot be the case, since we can produce a $p \times p$ lower-triangular matrix $A^{\triangle}_{\mathcal{M}}$, simply by adding the columns corresponding to the slack variables of the last $p - |\mathcal{M}|$ rows of $A_{\mathcal{M}}$. Hence, we conclude that $|\mathcal{M}| = p$.

2) Column-wise Greedy Search for the MTS: Algorithm 4 is a column-wise greedy search for the MTS. It successively adds the index of the maximum-weight degree-1 column of A to the set $\mathcal{M}$, and eliminates this column and the row that shares its only nonzero entry. Matrix A initially contains p degree-1 slack columns, and at each iteration, one such column will be erased. Hence, there is always a degree-1 column in the residual matrix, and the algorithm proceeds until p columns are selected. The resulting preconditioning set will correspond to an upper-triangular submatrix $A_{\mathcal{M}}$.

B. Implementation and Complexity Considerations

To compute the running time of Algorithm 3, note that while Step 4 has $O(q \log q)$ complexity, the computational complexity of the algorithm is dominated by the Triangulation Step. This subroutine has $O(q)$ complexity, and is called $O(q)$ times in Algorithm 3, which makes the overall complexity $O(q^2)$. An interesting problem to investigate in the future is whether we can simplify the triangulation process in line 7 to have sublinear complexity by exploiting the results of the previous round of triangulation.

[1]Note that if any set $\mathcal{S}$ of columns can be made lower triangular, any subset of these columns can be made lower triangular, as well.

Algorithm 4 Column-wise Greedy Search for the MTS

1: **Input:** $p \times q$ constraint matrix A, and the set of column weights $d_1, \dots, d_q$;
2: **Output:** A triangular set $\mathcal{M}$ and the upper-triangular matrix $A^{\triangle}_{\mathcal{M}}$;
3: **Initialization:** $\tilde{A} \leftarrow A$, $\mathcal{M} \leftarrow \emptyset$, and initialize col and row as zero-length vectors;
4: Define and form $\mathcal{DEG}1$ as the index set of all degree-1 columns in $\tilde{A}$;
5: **for** $k = 1$ to p **do**
6: $\quad$ Let $i \in \mathcal{DEG}1$ be the index of the (degree-1) column of $\tilde{A}$ with the maximum weight, d_i, and let j be the index of the row that contains the only nonzero entry of this column;
7: $\quad \mathcal{M} \leftarrow \mathcal{M} \cup i$, $col \leftarrow \begin{bmatrix} col \\ i \end{bmatrix}$, $row \leftarrow \begin{bmatrix} row \\ j \end{bmatrix}$;
8: $\quad$ Set all the entries in row j of $\tilde{A}$ (including the only nonzero entry of column i) to zero;
9: $\quad$ Update $\mathcal{DEG}1$ from the residual matrix, $\tilde{A}$;
10: **end for**
11: Form $A^{\triangle}_{\mathcal{M}}$ by setting $A^{\triangle}_{\mathcal{M}i,j} = A_{col_i, row_j}$, $\forall\ i, j \in \{1, \dots p\}$;

To assess the complexity of Algorithm 4, we need to examine Steps 4, 6, and 9, which deal with the list of degree-1 columns. One should be careful in selecting a suitable data structure for storing the set $\mathcal{DEG}1$, since, in each cycle of the for-loop, we need to extract the element with the maximum weight, and add to and remove from this set an $O(1)$ number of elements. By using a binary heap data structure [15], which is implementable as an array, all these (Steps 6 and 9) can be done in $O(\log q)$ time in the worst case. Also, the initial formation of the heap (Step 4) has $O(q)$ complexity. As a result, the complexity of Algorithm 4 becomes $O(q \log q)$.

The process of finding a triangular preconditioner is performed at each iteration of the IP algorithm. Since the values of primal variables, $\{x_i\}$, do not substantially change in one iteration, we expect the maximum-weight triangular set at each iteration to be relatively close to that in the previous iteration. Consequently, an interesting are for future work is to investigate modifications of the proposed algorithms, where the knowledge of the MTS in the previous iteration of the IP method is exploited to improve the complexity of these algorithms.

VI. Performance of the Preconditioners

A. Analytical Results

We will study the behavior of the proposed preconditioner in the later iterations of the IP algorithm, when the iterates are

close to the optimum. This is justified by the fact that, as the IP algorithm approaches the boundary of the feasible region during its later iterations, many of the primal variables, x_i, and the dual slack variables, z_i, approach zero, thus deteriorating the conditioning of the matrix $Q = AD^2A^T$. This is when a precoditioner is most needed.

Consider an LP problem in the augmented form (7) as part of ALP or MALP decoding, and assume that it has a unique optimal solution. We denote by the triple (x^*, y^*, z^*) the primal-dual solution to this LP, and by (x, y, z) an intermediate iterate of the IP method. We can partition the set of the q columns of A into the *basic set* $\mathcal{B}$ and the *nonbasic set* $\mathcal{N}$, where

$$\mathcal{B} = \{i | x_i^* > 0\} \quad \text{and} \quad \mathcal{N} = \{i | x_i^* = 0\}. \tag{15}$$

For any LP with p constraints, we have $|\mathcal{B}| \leq p$. In fact, the LPs solved for LP decoding are often "degenerate", i.e. $\mathcal{B} < p$.

It is known that the unique solution (x^*, y^*, z^*) is "strictly complementary" [16], meaning that for any $i \in \{1, \ldots, q\}$ either $x_i^* = 0$ and $z_i^* > 0$, or $x_i^* > 0$ and $z_i^* = 0$. Remembering from (10) that $d_i = \sqrt{x_i/z_i}$, as the iterates of the IP algorithm approach the optimum, i.e., μ given in (13) goes to zero, we will have

$$\lim_{\mu \to 0} d_i = \begin{cases} +\infty & \text{if } \ i \in \mathcal{B}, \\ 0 & \text{if } \ i \in \mathcal{N}, \end{cases} \tag{16}$$

Therefore, towards the end of the algorithm, the matrix $Q = AD^2A^T$ will be dominated by the columns of A and D corresponding to the basic set. Hence, it is highly desirable to select a preconditioning set that includes all the basic columns, i.e., $\mathcal{B} \subseteq \mathcal{M}$, in which case $A_{\mathcal{M}} D_{\mathcal{M}}^2 A_{\mathcal{M}}^T$ becomes a better and better approximation of Q, as we approach the optimum.

Lemma 1: Consider the constraint matrix A for an LP subproblem LP^k of MALP decoding, written in the augmented form (7) . If the primal solution to LP^k is integral, the set of columns of A corresponding to the basic variables form a matrix that can be made lower-triangular using column and row permutations.

Proof: Omitted. ∎

The following theorem shows that, under the conditions of Lemma 1, the incremental greedy Algorithm 3 indeed finds a preconditioning set that includes all the basic columns.

Theorem 3: Consider an LP subproblem LP^k of a MALP decoding problem. If the primal solution to LP^k is integral, at the iterates of the IP method that are sufficiently close to the solution, the Incremental Greedy Algorithm 3 finds a triangular set that includes all the columns corresponding to the basic set.

Proof: As the IP algorithm progresses, the basic variables approach 1, while the nonbasic variables approach 0. As a result of (16), after sufficient iterations, the $|\mathcal{B}|$ highest-weight columns of A will correspond to the basic set $\mathcal{B}$, and according to Lemma 1, the matrix $A_{\mathcal{B}}$ comprised of these columns can be made triangular. Algorithm 3 adds one of these columns to the preconditioning set at each iteration, until it includes all the $|\mathcal{B}|$ highest-weight (i.e., basic) columns. ∎

The assumption that the solution is integral does not hold for all LPs that we solve in adaptive LP decoding. However, in practice, we are often interested in solving the LP exactly only when the LP decoder has an integral solution (i.e., the ML codeword). This, of course, does not necessarily mean that in such cases every LP subproblem solved in the adaptive method has an integral solution. On the other hand, one would intuitively expect that, if the final LP subproblem has an integral solution, the intermediate LPs are also very likely to have an integral solution, since the factor graphs corresponding to the intermediate problems are subgraphs of the Tanner graph of the code, and thus generally contain fewer cycles.

While we studied the IP method in the context of MALP decoding, the proposed algorithms can also be applied to the LPs that may have more than one constraint from each check node, such as those arising in ALP decoding. However, in the absence of the single-constraint property, some of the analytical results we presented may no longer be valid.

B. Simulation Results

We simulated LP decoding using the MALP algorithm and our sparse implementation of the path-following IP method on the AWGN channel. We used a randomly-generated $(3, 6)$-regular LDPC code of length 2000, with the 4-cycles removed from its Tanner graph. We have shown before that, as the IP algorithm progresses, the matrix AD^2A^T that needs to be inverted to compute the Newton steps becomes more and more ill-conditioned. We have observed that this problem becomes more severe for LP problems in the later iterations of the MALP algorithm, where the LP is larger and more degenerate due to the abundance of active constraints at the solution to the problem.

In Figs. 1 and 2, we present the performance results of the PCG method for two different systems of linear equations in the form of (9), solved by the infeasible primal-dual path-following IP algorithm, using the preconditioners designed by greedy Algorithms 3 and 4. The performance of the preconditioned CG (PCG) algorithm is measured by the behavior of the relative residual error $\|Qx - w\|^2 / \|w\|^2$ in (11), where $\|.\|$ denotes the Euclidian norm, as a function of the iteration number of the PCG algorithm.

In Fig. 1, we considered solving (9) using the PCG method in the 18th iteration of the IP algorithm. The LP problem was selected from the 6th iteration of a MALP decoding problem at SNR = 1.5 dB, and the solution to the LP was *integral*. The constraint matrix A for this LP had 713 rows and 2713 columns. In this scenario, the duality gap $g_d = x^T z$ was equal to 0.22, and the condition number of the problem, i.e., $\kappa(Q)$, was equal to 2.33×10^8. We have plotted the residual error of the CG method without preconditioning, as well as the PCG method using the two proposed preconditioner designs. In this problem, the convergence of the CG methods is very slow, so that in 200 iterations, the residual error does not get below 0.07. On the other hand, the PCG method with incremental greedy preconditioning has a very fast convergence, reaching

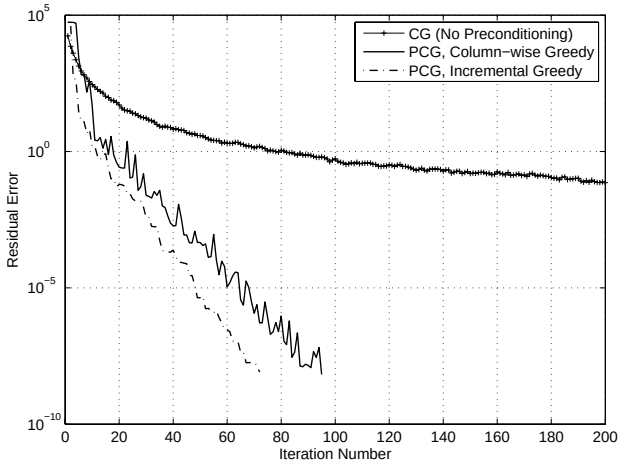

Fig. 1. The residual error for different PCG implementations, solving (9) in the 18th iteration of the IP algorithm, in an LP with an integral solution.

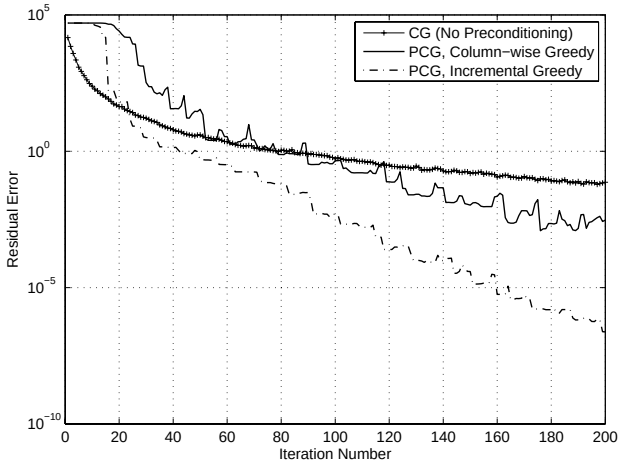

Fig. 2. The residual error for different PCG implementations, solving (9) in the 18th iteration of the IP algorithm, in an LP with a fractional solution.

a residual error of 10^{-4} in 40 iterations, closely followed by the column-wise greedy preconditioner.

In Fig. 2, we considered an LP from the 6th iteration of a MALP decoding problem at SNR = 1.0 dB, where the solution was *fractional*. The matrix A had 830 rows and 3830 columns, and we compute the Newton step at the 18th (penultimate) iteration of the IP algorithm, with $g_d = 0.155$ and $\kappa(Q) = 2.61 \times 10^8$. These parameters are chosen such that the scenario in this figure is similar to the one in Fig. 1, the main difference being that the decoding problem now has a fractional solution. We observe that, while the performance of the CG method is very similar in Fig. 1 and Fig. 2, the preconditioned methods have slower convergence when the LP solution is fractional.

VII. Conclusion

We studied a sparse interior-point (IP) implementation of LP decoding, with the goal of exploiting the properties of the decoding problem in order to achieve lower complexity. To that end, we first proposed a modification of the adaptive LP decoding algorithm, which has the single-constraint property; i.e., it solves a sequence of LPs that each contain at most one parity inequality from each parity check.

The heart of the IP algorithm is the computation of the Newton step at each iteration via solving a system of linear equations. These systems of equations are often ill-conditioned, especially in the later iterations. In such cases, the iterative algorithms for solving sparse linear systems, including the conjugate-gradient method, convergence slowly. Motivated by the properties of LP decoding, we studied a new framework for desiging a preconditioner. Our approach was based on finding a matrix that approximates the constraint matrix, and in addition, can be inverted in linear time due to its combinatorial structure. We proposed two greedy algorithms for designing such preconditioners, and showed that, when the solution to the LP is integral, one of these algorithms can provably find a preconditioner that is a good approximation of the original matrix. Lastly, we demonstrated the performance of the proposed schemes via simulation, and we observed that the preconditioned systems are most effective when the LP has an integral solution.

Acknowledgment

This work is supported in part by NSF Grant CCF-0829865.

References

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

[2] M. H. Taghavi and P. H. Siegel, "Adaptive linear programming decoding," *Proc. IEEE Int'l Symp. on Inform. Theory*, Seattle, WA, Jul. 2006.

[3] M. Chertkov and M. Stepanov, "Pseudo-codeword landscape," *Proc. IEEE Int'l Symp. on Inform. Theory, ISIT'07*, Nice, France, Jun. 2007.

[4] K. Yang, X. Wang, and J. Feldman, "Cascaded formulation of the fundamental polytope of general linear block codes," *Proc. IEEE Int'l Symp. on Inform. Theory, ISIT'07*, Nice, France, Jun. 2007.

[5] P. O. Vontobel, "Interior-point algorithms for linear-programming decoding," *Proc. Information Theory and its Applications Workshop*, La Jolla, CA, Jan./Feb. 2008.

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

[7] P. O. Vontobel and R. Koetter, "Towards low-complexity linear-programming decoding," *Proc. Int'l Conf. on Turbo Codes and Related Topics*, Munich, Germany, Apr. 2006.

[8] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," *Sumbitted to IEEE Trans. Inform. Theory*.

[9] M. H. Taghavi and P. H. Siegel, "Adaptive methods for linear programming decoding," *To appear in IEEE Trans. Inform. Theory*.

[10] M. H. Taghavi, A. Shokrollahi, and P. H. Siegel, "Efficient interior-point implementation of linear programming decoding," *Submitted to IEEE Trans. Inform. Theory*.

[11] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*,, Athena Scientific, Belmont, MA, 1997.

[12] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409-436, Dec. 1952.

[13] Y. Saad, *Iterative Methods for Sparse Linear Systems (2nd Edition)*. SIAM, 2003.

[14] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 638-656, Feb. 2001.

[15] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed. Reading, Addison-Wesley, MA, 1998.

[16] A. J. Goldman and A. W. Tucker, "Theory of linear programming," *Linear Equalities and Related Systems*, H. W. Kuhn and A. W. Tucker, eds., Princeton University Press, Princeton, N. J., 1956, pp. 53-94.