ISIT'2013 1569718969

Efficient Iterative LP Decoding of LDPC Codes with Alternating Direction Method of Multipliers

Xiaojie Zhang
Samsung R&D America, Dallas, Texas 75082
Email: eric.zhang@samsung.com

Paul H. Siegel
University of California, San Diego, CA 92093
Email: psiegel@ucsd.edu

***Abstract*—In this paper, we propose an efficient message-passing algorithm to solve the LP decoding problem. This algorithm is based on the alternating direction method of multipliers (ADMM), a classic technique in convex optimization theory that is designed for parallel implementation. The computational complexity of ADMM-based LP decoding is largely determined by the method used to project a vector of real values to the parity polytope of a given parity check. The key contribution of this paper is a novel, efficient projection algorithm that can substantially improve the decoding speed of the ADMM-based LP decoder.**

I. Introduction

Alternating direction method of multipliers (ADMM) is a classic optimization technique that was developed in the 1970s, which combines the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization [1]. The use of ADMM in LP decoding was first suggested by Barman *et al.* [2]. The LP problem in decoding LDPC codes is a constrained convex optimization problem, which is readily solved by ADMM techniques [1, Chapter 5]. A key component in solving constrained optimization problems with ADMM is the method used to project a vector of real values to the convex set of feasible solutions, known as the fundamental polytope in the context of LP decoding [5]. In [2], Barman *et al.* proposed a projection algorithm, which was further improved in [3]. This algorithm is based on the fact that a constrained convex polytope defined by a parity check can be expressed as the convex hull of a set of "slices," where each slice is the convex hull of all binary vectors of the same even weight. Hence, for a given vector, the algorithm first sorts the coordinates of a vector in descending order and characterizes two slices whose convex hull contains the projection; then, it finds the projection of the ordered vector by solving a quadratic program. However, the ordering of all coordinates that is required before projecting any given vector increases the complexity of the algorithm.

In this paper, we propose a novel and efficient projection algorithm for ADMM-based LP decoding. Based on the cut search algorithm (CSA) introduced in [4], the proposed algorithm can efficiently determine whether a vector is inside the check polytope without first ordering its coordinates. At the same time, if the vector lies outside the polytope, the algorithm identifies the hyperplane containing the projection. Our software implementation of ADMM-based LP decoding demonstrates that the proposed projection algorithm is more computationally efficient than the "two-slice" projection algorithm proposed in [2] and [3].

The remainder of the paper is organized as follows. In Section II, we review the formulation of LP decoding and its ADMM presentation. In Section III, we describe the proposed projection algorithm. Section IV presents some complexity analysis and numerical results, and Section V concludes the paper.

II. Formulation of LP Decoding

A. LP Relaxation of ML Decoding

Consider a binary linear block code $\mathcal{C}$ of length n and a corresponding $m \times n$ parity-check matrix $\mathbf{H}$. Each row of $\mathbf{H}$ corresponds to a parity check, or check node in the Tanner graph, indexed by $J = \{1, \ldots, m\}$, and each column corresponds to a codeword symbol, or variable node in the Tanner graph, indexed by $I = \{1, \ldots, n\}$. Let $N_j = \{i \in I : H_{ji} = 1\}$ be the index set of neighboring variables of check $j \in J$, and analogously let $N_i = \{j \in J : H_{ji} = 1\}$ be the index set of neighboring checks of variable $i \in I$. The degree of check j is denoted by $d_j = |N_j|$, and the degree of variable i is $d_i = |N_i|$, where for a set $\mathcal{X}$, $|\mathcal{X}|$ denotes its cardinality. Finally, let $N_j(k) \in N_j$ be the kth element in N_j, $1 \le k \le |N_j|$.

A codeword $\mathbf{y} \in \mathcal{C}$ is transmitted across a memoryless binary-input output-symmetric (MBIOS) channel, resulting in a received vector $\mathbf{r}$. Assuming that the transmitted codewords are equiprobable, the ML decoder finds the solution to the following optimization problem

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

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

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

The transfer matrix $\mathbf{T}_j$ selects from an n-vector the coordinates corresponding to the d_j neighboring variables of check j. For example, if the jth row of parity-check matrix $\mathbf{H}$ is $\mathbf{h}_j = (0, 1, 0, 1, 0, 1, 0)$, then the corresponding transfer matrix is

$$\mathbf{T}_j = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Hence, $\mathbf{T}_j\mathbf{x}$ has dimension $|N_j|$.

Definition 1: The *check polytope*, $\mathcal{P}_d$, is the convex hull of all binary vectors of length d with an even number of 1s, i.e.,

$$\mathcal{P}_d \triangleq \text{ conv}\left(\left\{\mathbf{x} \in \{0,1\}^d \mid \mathbf{x} \text{ has an even number of 1s}\right\}\right).$$

Note that the definition of the check polytope resembles that of the *parity polytope* [6], the difference being that, in the latter, the number of 1s in $\mathbf{x}$ is odd.

Feldman *et al.* [5] shown that the ML decoding problem in (1) can be relaxed to the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\gamma}^T\mathbf{u} \\ \text{subject to} \quad & \mathbf{T}_j\mathbf{u} \in \mathcal{P}_{d_j} \quad \forall j \in J. \end{aligned} \tag{2}$$

B. ADMM formulation

One way to solve the optimization problem (2) is to use ADMM, which is intended to combine the decomposability of dual ascent with the superior convergence properties of the method of multipliers [1]. In order to make the LP decoding problem perfectly fit the ADMM template given in [1, p. 33], we first rewrite (2) as follows

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{\gamma}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{T}_j\mathbf{x} = \mathbf{z}_j,\ \mathbf{z}_j \in \mathcal{P}_{d_j},\ \forall j \in J. \end{aligned} \tag{3}$$

The augmented Lagrangian (using the scaled dual variable) is then

$$L_\rho(\mathbf{x},\mathbf{z},\mathbf{y}) = \boldsymbol{\gamma}^T\mathbf{x} + \frac{\rho}{2}\sum_{j\in J}\|\mathbf{T}_j\mathbf{x} - \mathbf{z}_j + \mathbf{y}_j\|_2^2 - \frac{\rho}{2}\sum_{j\in J}\|\mathbf{y}_j\|_2^2, \tag{4}$$

where $\mathbf{y}_j \in \mathbb{R}^{d_j}$ is the *scaled dual variable* and $\rho > 0$ is called the *penalty parameter*.

So, the scaled form of ADMM for this problem consists of the following iterations

$$\mathbf{x}^{k+1} := \underset{\mathbf{x}}{\text{argmin}}\left(\boldsymbol{\gamma}^T\mathbf{x} + \frac{\rho}{2}\sum_{j\in J}\|\mathbf{T}_j\mathbf{x} - \mathbf{z}_j^k + \mathbf{y}_j^k\|_2^2\right) \tag{5}$$

$$\mathbf{z}_j^{k+1} := \Pi_{\mathcal{P}_{d_j}}\left(\mathbf{T}_j\mathbf{x}^{k+1} + \mathbf{y}_j^k\right) \tag{6}$$

$$\mathbf{y}_j^{k+1} := \mathbf{y}_j^k + \mathbf{T}_j\mathbf{x}^{k+1} - \mathbf{z}_j^{k+1}, \tag{7}$$

where $\Pi_{\mathcal{P}_d}(\mathbf{u})$ is the Euclidean projection of vector $\mathbf{u}$ on $\mathcal{P}_d$.

After some simplification, the ADMM-based decoding algorithm can be expressed in the form of an iterative message-passing algorithm, as described in Algorithm 1. Each check j updates its outgoing messages based on the received messages from its neighboring variables (i.e., x_i for $i \in N_j$), and its locally stored $\mathbf{y}_j$ from the previous iteration. For each variable, the variable update computation uses only the incoming messages from neighboring checks, as can be seen in (10). As with other message-passing decoding algorithms, one can compute the updates for all the checks simultaneously, and likewise for all the variables.

In Step 1 of Algorithm 1, there can be different ways to initialize $\mathbf{x}$. For example, we can set

$$x_i = \begin{cases} 0 & \text{if } \gamma_i \geq 0 \\ 1 & \text{if } \gamma_i < 0. \end{cases} \tag{11}$$

Algorithm 1 Iterative ADMM-based LP decoding algorithm

1: **Initialization:** Properly initialize $\mathbf{x}$, $\mathbf{z}_j$, and $\mathbf{y}_j$ $\forall j \in J$.

2: **Check update:** For each check $j \in J$, update $\mathbf{z}_j$ and $\mathbf{y}_j$ using (6) and (7), respectively. We rewrite them as

$$\mathbf{w} \leftarrow \mathbf{T}_j\mathbf{x} + \mathbf{y}_j \quad \mathbf{z}_j \leftarrow \Pi_{\mathcal{P}_{d_j}}(\mathbf{w}) \quad \mathbf{y}_j \leftarrow \mathbf{w} - \mathbf{z}_j. \tag{8}$$

The message transmitted to neighboring variables is

$$L_{j\to i} \leftarrow (z_j)_i - (y_j)_i. \tag{9}$$

3: **Variable update:** For each variable $i \in I$, the messages transmitted to its neighboring checks are the same, and are computed using (5), which can be rewritten as

$$x_i \leftarrow \frac{1}{d_i}\left(\sum_{j'\in N_i} L_{j'\to i} - \frac{\gamma_i}{\rho}\right). \tag{10}$$

4: **Stopping criterion:** If $\sum_{j\in J}\|\mathbf{T}_j\mathbf{x} - \mathbf{z}_j\|_2 < \epsilon^{\text{pri}}$ and $\sum_{j\in J}\|\mathbf{z}_j^k - \mathbf{z}_j^{k-1}\|_2 < \epsilon^{\text{dual}}$, where $\epsilon^{\text{pri}} > 0$ and $\epsilon^{\text{dual}} > 0$ are feasibility tolerances, the ADMM algorithm converges and $\mathbf{x}$ is the optimal solution; otherwise, go to Step 2.

If this $\mathbf{x}$ is not a valid codeword, then one starts the ADMM decoding; otherwise, it is the ML codeword. The scaled dual variable $\mathbf{y}_j$ can be set to be all zeros for all $j \in J$. The initialization of $\mathbf{z}_j$ does not affect the ADMM decoding. Note that different initializations of $\mathbf{x}$, $\mathbf{z}_j$, and $\mathbf{y}_j$ can only affect the number of iterations required to converge, and they would not change the optimal solution obtained by ADMM (within the feasibility tolerances).

In Algorithm 1, except for the projection $\Pi_{\mathcal{P}_{d_j}}$ in (8), the computations are quite straightforward, involving only additions and multiplications, and are standard update procedures of ADMM taken from [1]. However, the projection $\Pi_{\mathcal{P}_{d_j}}$ of a given vector onto the check polytope of dimension d_j has to be specially designed, and the efficiency of the projection algorithm directly determines the complexity of ADMM decoding. In the next section, we will introduce the key contribution in this paper, which is an efficient algorithm that projects any given vector onto the check polytope.

III. Efficient Projection onto Check Polytope

In [2] and [3], two projection algorithms were proposed. They are based on the so-called "two-slice" representation of any vector in the check polytope, whereby any given vector can be expressed as a convex combination of two binary vectors of Hamming weight r and $r+2$, for some even integer r. The Majorization Theorem [7] is used to characterize the convex hull of the two slices, where a sorting of all coordinates of the given vector is required to decide if it is within the check polytope. If the given vector is outside the polytope, the projection algorithm involves two steps: the vector is first projected onto a scaled version of one of the two slices, and then the residual is projected onto another scaled slice [2]. Find the appropriate scaling value is difficult, however. In [3],

this issue is addressed by expressing the projection problem as quadratic program after the two slices have been characterized by the majorization method.

In this section, we propose a new, more efficient projection algorithm. We first use the cut search algorithm (CSA), introduced in [4], to determine the facet of the polytope in which the projection point lies. We then use an efficient algorithm to project the vector onto this facet.

A. Cut Search Algorithm

From [5], we know that the check polytope $\mathcal{P}_d$ of dimension d can be described by a set of box constraints and parity inequalities as follows

$$0 \leq u_i \leq 1, \ \forall i \in [d] \tag{12}$$

$$\sum_{i \in V} u_i - \sum_{i \in V^c} u_i \leq |V| - 1, \ \forall V \subseteq [d] \text{ with } |V| \text{ odd}. \tag{13}$$

Define by $\boldsymbol{\theta}_V$ the *indicator vector* of set V such that

$$\theta_{V,i} = \begin{cases} 1 & \text{if } i \in V \\ -1 & \text{if } i \in V^c, \end{cases}$$

where $V^c = [d] \setminus V$ is the complement of V. Then, the inequality (13) can be written as

$$\boldsymbol{\theta}_V^T \mathbf{u} \leq |V| - 1.$$

For a given vector $\mathbf{u} \in [0,1]^d$, if there exists a set $V \in [d]$ of odd cardinality such that

$$\boldsymbol{\theta}_V^T \mathbf{u} > |V| - 1, \tag{14}$$

then $\mathbf{u}$ lies outside the check polytope. The violated parity inequality (14) is called a *cut* at $\mathbf{u}$, and the corresponding V of odd cardinality is called a *cutting set*. The following result shows that, for a given vector, there exists at most one cutting set.

Proposition 1 (Th. 1 in [8]): If at any given point $\mathbf{u} \in [0,1]^d$, one of the parity inequalities in (13) is violated, the rest of them are satisfied with strict inequality.

In [4], Proposition 1 served as the motivation for a highly efficient technique for finding cuts, the Cut Search Algorithm (CSA), was introduced. The CSA is shown in Algorithm 2, from which it is clear that, to determine whether a given vector lies inside the check polytope or not, the algorithm visits every coordinate only once, and no ordering of the coordinates is required.

B. Projection onto the Check Polytope

It can be seen from Definition 1 that the check polytope $\mathcal{P}_d$ lies inside the $[0,1]^d$ hypercube, i.e., $\mathcal{P}_d \subset [0,1]^d$. This means that all points outside the hypercube are also outside the check polytope. To find the check polytope projection of a vector outside the hypercube, we first check with CSA whether or not its projection onto the hypercube is also in the check polytope. If the hypercube projection is outside the check polytope, the cut found by CSA can be further used to find the check polytope projection, as will be shown later in this subsection.

Algorithm 2 Cut search algorithm

Input: vector $\mathbf{u} \in [0,1]^d$.

Output: indicator vector $\boldsymbol{\theta} \in \{-1,1\}^d$ of cutting set V (if exists)

1: $\theta_i \leftarrow \begin{cases} 1 & \text{if } u_i > \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}$
2: **if** the cardinality of set $\{i : \theta_i > 0\}$ is even **then**
3: $\quad i^* \leftarrow \underset{i}{\operatorname{argmin}} \left|\frac{1}{2} - u_i\right|$.
4: $\quad \theta_{i^*} \leftarrow -\theta_{i^*}$.
5: **end if**
6: $V = \{i : \theta_i > 0\}$.
7: **if** $\boldsymbol{\theta}^T \mathbf{u} > |V| - 1$ **then**
8: $\quad$ **return** $\boldsymbol{\theta}$ is the indicator vector of cutting set V, and $\mathbf{u} \notin \mathcal{P}_d$.
9: **else**
10: $\quad$ **return** No cut found, and $\mathbf{u} \in \mathcal{P}_d$.
11: **end if**

The projection of any vector onto the hypercube can be done easily by applying a threshold on all coordinates. Let $\mathbf{z} = \Pi_{[0,1]^d}(\mathbf{u})$ be the $[0,1]^d$ projection of vector $\mathbf{u} \in \mathbb{R}^d$; then we have

$$z_i = \begin{cases} 1 & \text{if } u_i > 1 \\ 0 & \text{if } u_i < 0 \\ u_i & \text{otherwise.} \end{cases} \tag{15}$$

It is easy to verify that the $\mathbf{z}$ computed in (15) is indeed the Euclidean projection of $\mathbf{u}$ onto the $[0,1]^d$ hypercube. If $\mathbf{z} \in \mathcal{P}_d$, then $\mathbf{z}$ is exactly $\Pi_{\mathcal{P}_d}(\mathbf{u})$, and the projection is completed. Therefore, in the remaining part of this subsection, we focus on vectors whose $[0,1]^d$ projection $\mathbf{z} = \Pi_{[0,1]^d}(\mathbf{u})$ is outside the check polytope, i.e., $\mathbf{z} \notin \mathcal{P}_d$. From Proposition 1, there exists only one set V of odd cardinality such that $\boldsymbol{\theta}_V^T \mathbf{z} > |V| - 1$. We will make use of the following relationship between $\mathbf{u}$ and its $[0,1]^d$ projection, $\mathbf{z}$.

Proposition 2: Given a vector $\mathbf{u} \in \mathbb{R}^d$, let $\mathbf{z} = \Pi_{[0,1]^d}(\mathbf{u})$. If there exists a cut $\boldsymbol{\theta}_V^T \mathbf{z} > |V| - 1$, then $u_i \geq z_i$ for all $i \in V$ and $u_i \leq z_i$ for all $i \in V^c$. This implies $\boldsymbol{\theta}_V^T \mathbf{u} \geq \boldsymbol{\theta}_V^T \mathbf{z}$, where equality holds if and only if $\mathbf{z} = \mathbf{u}$.

The proposed projection algorithm is based on the following result.

Theorem 3: For a given vector $\mathbf{u} \in \mathbb{R}^d$, let $\mathbf{z} = \Pi_{[0,1]^d}(\mathbf{u})$. If V is a cutting set for $\mathbf{z}$, i.e., $\boldsymbol{\theta}_V^T \mathbf{z} > |V| - 1$, then $\Pi_{\mathcal{P}_d}(\mathbf{u})$ must be on the facet of $\mathcal{P}_d$ defined by V, i.e., $\Pi_{\mathcal{P}_d}(\mathbf{u}) \in \mathcal{F}_V \triangleq \left\{\mathbf{x} \in [0,1]^d \mid \boldsymbol{\theta}_V^T \mathbf{x} = |V| - 1\right\}$.

Proof: Proposition 1 implies that the cutting set V for $\mathbf{z}$ is unique and that for any vector $\mathbf{x} \in \mathcal{F}_V \cap [0,1]^d$, $\mathbf{x} \in \mathcal{P}_d$. Suppose that $\Pi_{\mathcal{P}_d}(\mathbf{u}) = \mathbf{w} \in \mathcal{P}_d$, but $\mathbf{w} \notin \mathcal{F}_V$; then, $\mathbf{w}$ must satisfy $\boldsymbol{\theta}_V^T \mathbf{w} < |V| - 1$.

Let $s = \boldsymbol{\theta}_V^T \mathbf{z} - |V| + 1 > 0$, $t = |V| - 1 - \boldsymbol{\theta}_V^T \mathbf{w} > 0$, and define vector $\mathbf{v} = \frac{t}{s+t}\mathbf{z} + \frac{s}{s+t}\mathbf{w}$. Then,

$$\boldsymbol{\theta}_V^T \mathbf{v} = \frac{t}{s+t}\boldsymbol{\theta}_V^T \mathbf{z} + \frac{s}{s+t}\boldsymbol{\theta}_V^T \mathbf{w} = |V| - 1,$$

which implies $\mathbf{v} \in \mathcal{F}_V$. The vector $\mathbf{v}$ is also in the convex hypercube $[0,1]^d$ because it is a linear combination of $\mathbf{z}$ and $\mathbf{w}$. Hence, by Proposition 1, we have $\mathbf{v} \in \mathcal{P}_d$.

Since $\mathbf{z}$ is the projection of $\mathbf{u}$ onto the $[0,1]^d$ hypercube, then it follows from (15) that for any $x \in [0,1]^d$, we have the following equality

$$|u_i - x_i| = |u_i - z_i| + |z_i - x_i|, \quad 1 \le i \le d. \tag{16}$$

From the definition of $\mathbf{v}$, we have

$$z_i - v_i = \frac{s}{s+t}(z_i - w_i), \quad 1 \le i \le d. \tag{17}$$

Setting $\mathbf{x} = \mathbf{v}$ and substituting (17) into (16), we get

$$\begin{aligned} |u_i - v_i| &= |u_i - z_i| + \frac{s}{s+t}|z_i - w_i| \\ &\le |u_i - z_i| + |z_i - w_i| \\ &= |u_i - w_i|. \end{aligned}$$

Since $\mathbf{z} \neq \mathbf{w}$, the inequality above cannot hold with equality for all i, hence $\|\mathbf{u} - \mathbf{v}\|_2^2 < \|\mathbf{u} - \mathbf{w}\|_2^2$. Because the check polytope is convex, the Euclidean projection of any vector onto it is unique. This contradicts the assumption that $\mathbf{w}$ is the Euclidean projection of $\mathbf{u}$ onto $\mathcal{P}_d$. ∎

Now the only thing needed to complete the projection is an efficient algorithm to solve the following optimization problem: for any given vector $\mathbf{u} \notin \mathcal{P}^d$,

$$\begin{aligned} \text{minimize} \quad & \|\mathbf{u} - \mathbf{x}\|_2^2 \\ \text{subject to} \quad & 0 \le x_i \le 1, \quad i = 1, \ldots, d \\ & \boldsymbol{\theta}_V^T \mathbf{x} = |V| - 1, \end{aligned} \tag{18}$$

Since the optimization problem (18) has differentiable convex objective and constraint functions, any points that satisfy the Karush-Kuhn-Tucker (KKT) conditions are optimal solutions. With some algebra, omitted due to space limitations, the solution to (18) can be expressed as

$$\mathbf{x}^* = \Pi_{[0,1]^d}\left(\mathbf{u} - \nu^* \boldsymbol{\theta}_V\right), \tag{19}$$

where ν^* is a scalar such that $\boldsymbol{\theta}_V \mathbf{x}^* = |V| - 1$.

Algorithm 3 describes an efficient method for computing the scalar ν^*. Its complexity is $|T| \log(|T|$, where $|T|$ is usually much smaller than d. By combining Algorithms 2 and 3, we can determine the projection of any given point onto the check polytope, as described in Algorithm 4. This algorithm can be incorporated into Step 2 (check update) in the ADMM-base LP decoder, Algorithm 1.

In terms of computational efficiency, Algorithm 4 compares favorably to the two-slice projection algorithms proposed in [2] and [3]. In the two-slice algorithms, the determination of whether or not a given point lies inside the check polytope requires that the coordinates of the point be ranked in descending order. In contrast, the proposed algorithm finds the cutting facet on which the projection of the point lies by visiting each coordinate only once. For a point outside the check polytope, the two-slice algorithms represent the projection in a form similar to that in (19). However, the underlying optimization problem that they solve requires rank ordering of a set of size $2d + 2$ and may also need to check all of the elements in the set before finding the optimal scalar ν^* [3, Algorithm. 2]. In contrast, the proposed Algorithm 3 only considers the coordinates whose values lie outside the interval $[0,1]$, and the calculation which uses them is relatively much simpler. As we will show in Section IV, with the same ADMM parameters, the iterative ADMM-based LP decoder incorporating the proposed algorithm runs at least 3 times faster than the one based upon the improved two-slice projection algorithm in [3].

Algorithm 3 Solving optimization problem (18)

Input: vector $\mathbf{u} \in \mathbb{R}^d$ and vector $\boldsymbol{\theta}_V \in \{-1, 1\}^d$.
Output: solution of optimization problem (18) in terms of ν^*.

1: $T \leftarrow \{u_i - 1 \mid u_i > 1\} \cup \{-u_i \mid u_i < 0\}$, $\delta \leftarrow \boldsymbol{\theta}_V^T \mathbf{u} - |V| + 1$, and $\zeta = d$.
2: **if** $T \neq \emptyset$ **then**
3: Sort elements in $T = \{t_i\}$ such that $t_i \ge t_{i+1}$.
4: **for** $i = 1$ to $|T|$ **do**
5: **if** $\delta/\zeta > t_i$ **then**
6: Exit to line 12.
7: **else**
8: $\delta \leftarrow \delta - t_i$ and $\zeta \leftarrow \zeta - 1$.
9: **end if**
10: **end for**
11: **end if**
12: **return** $\nu^* \leftarrow \delta/\zeta$ is optimal solution.

Algorithm 4 Efficient check polytope projection algorithm

Input: vector $\mathbf{u} \in \mathbb{R}^d$.
Output: the projection $\Pi_{\mathcal{P}_d}(\mathbf{u}) \in \mathcal{P}_d$.

1: Run Algorithm 2 with input $\Pi_{[0,1]^d}(\mathbf{u})$.
2: **if** cutting set V is found **then**
3: Run Algorithm 3 with input parameters $\mathbf{u}$ and $\boldsymbol{\theta}_V$, then get output ν^*.
4: **return** $\Pi_{[0,1]^d}(\mathbf{u} - \nu^* \boldsymbol{\theta}_V)$.
5: **else**
6: **return** $\Pi_{[0,1]^d}(\mathbf{u})$.
7: **end if**

IV. Numerical Results

To demonstrate the improved performance offered by the proposed ADMM-based LP decoder, we compare its performance to that of several other LDPC decoders, such as an ADMM-based LP decoder using the two-slice projection algorithm [3], an adaptive LP (ALP) decoder [9], and a floating-point box-plus SPA decoder [11]. We apply these decoders to two LDPC codes of different lengths, rates, and degree distributions, namely a rate-0.77, (3,13)-regular LDPC code of length 1057 [10] and a rate-$\frac{1}{3}$ irregular LDPC code of length 1920 [10]. The frame error rate (FER) curves are based on Monte Carlo simulations that generated at least 200 error frames for each point in the plots.

The performance of the ALP decoder can be viewed as the

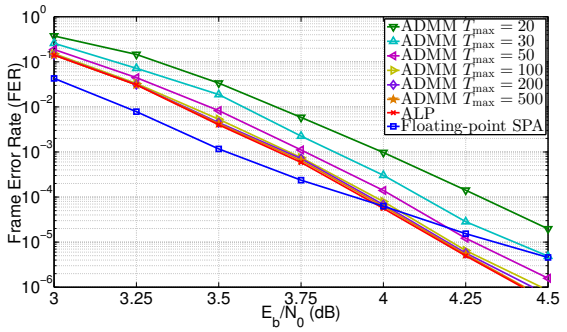
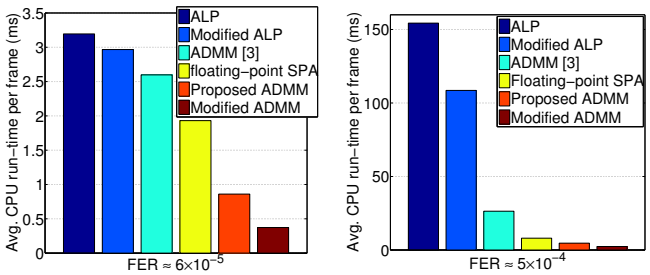


Fig. 1. FER comparison of ADMM-base LP decoder with various maximum number of iterations for (1057,833) LDPC code on AWGNC.



(a) Rate 0.77 and length 1057. (b) Rate 0.33 and length 1920.

Fig. 2. Average simulation time for decoding one codeword.

best that ADMM-based LP decoders can achieve by solving the standard LP decoding problem (3). It can be seen that the error-rate performance of ADMM-based LP decoding with $T_{\max} = 100$ is already very close to that of ALP decoding, and with $T_{\max} = 200$, it essentially achieves the ALP decoder performance. We can also see that the error-rate curves of all of the LP decoders are steeper than that of the SPA decoder. (A similar phenomenon was observed in [4].)

Fig. 2 presents an intuitive way of comparing the computational complexity of different decoding algorithms. It compares the average decoding time when we implement these algorithms using C++ code on a desktop PC. The SPA decoder is implemented in software with messages represented as double-precision floating-point numbers, and the pair-wise box-plus SPA is computed in the manner described in [11]. The SPA decoder iterations stop as soon as a codeword is found, or when the maximum allowable number of iterations $T_{\max} = 200$ have been attempted. To ensure a fair comparison with the two-slice projection algorithm in [2], we incorporated into our C++ platform for ADMM-based LP decoders the C++ code obtained from the website [12] of one of the authors of [2].

The simulation time is averaged over one million frames for each decoder, and the channel condition is set to let all decoders achieve approximately the same FER as shown in the figures. We can see that, compared to the two-slice projection algorithm in [3], the proposed projection algorithm significantly improves the efficiency of the ADMM-based LP decoder, especially for low-rate codes. The modified ALP (MALP) decoding algorithm [9], which is much faster than the standard LP decoder proposed by Feldman *et al.* in [5], uses the open-source GNU Linear Programming Kit (GLPK) as its LP solver. From the simulation results, we can see that all ADMM-based LP decoders are faster than MALP, especially for the low-rate code, where the decoding time is reduced by more than 3 orders of magnitude. This is because the general-purpose LP solver used in MALP does not take advantage of the sparsity of the constraints in the LP decoding of LDPC codes. The modified ADMM uses early termination and over-relaxation to further improve the decoding speed by a factor of 2. Details can be found in [13].

V. Conclusion

In this paper, we propose an efficient message-passing algorithm to solve the LP decoding problem based on ADMM. The computational complexity of ADMM-based LP decoding is largely determined by the method used to project a vector of real values to the check polytope of a given parity check. We proposed a novel, efficient projection algorithm that can substantially improve the decoding speed of the ADMM-based LP decoder.

Acknowledgment

This research was supported in part by the Center for Magnetic Recording Research at UCSD, by UC Lab Fees Research Program, Award No. 09-LR-06-118620-SIEP, and by NSF Grant CCF-1116739.

References

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[2] S. Barman, X. Liu, S. Draper, and B. Recht, "Decomposition methods for large scale LP decoding," *Proc. 46th Allerton Conf. Commun., Control, Computing*, Monticello, IL, Sep. 2011, pp. 253–260.

[3] S. Barman, X. Liu, S. Draper, and B. Recht, "Decomposition methods for large scale LP decoding," arXiv:1204.0556 [cs.IT], 2012.

[4] X. Zhang and P. Siegel, "Adaptive cut generation algorithm for improved linear programming decoding of binary linear codes," *IEEE Trans. Inform. Theory*, vol. 58, no. 10, pp. 6581–6594, Oct. 2012.

[5] J. Feldman, M. 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.

[6] M. Yannakakis, "Expressing combinatorial optimization problems by linear programs," *J. Computer and System Sciences*, vol. 43, no. 3, pp. 441–466, 1991.

[7] A. Marshall, I. Olkin, and B. Arnold, *Inequalities: Theory of Majorization and its Applications*. New York: Springer, 2010.

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

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

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

[11] X. Hu, E. Eleftheriou, D. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *Proc. IEEE Globecom*, San Antonio, Nov. 2001, pp. 1036–1036E.

[12] C++ ADMM Decoder by X. Liu. [Online]. Available: https://sites.google.com/site/xishuoliu/codes

[13] X. Zhang and P. Siegel, "Efficient iterative LP decoding of LDPC codes with Alternating Direction Method of Multipliers," to be submitted to *IEEE Trans. Inform. Theory*.