Decomposition Methods for Large Scale LP Decoding

Siddharth Barman* Xishuo Liu† Stark Draper‡ Benjamin Recht§

Abstract

Feldman *et al.* (*IEEE Trans. Inform. Theory*, Mar. 2005) showed that linear programming (LP) can be used to decode linear error correcting codes. The bit-error-rate performance of LP decoding is comparable to state-of-the-art BP decoders, but has significantly stronger theoretical guarantees. However, LP decoding when implemented with standard LP solvers does not easily scale to the block lengths of modern error correcting codes. In this paper we draw on decomposition methods from optimization theory to develop efficient distributed algorithms for LP decoding. The key enabling technical result is a nearly linear time algorithm for two-norm projection onto the parity polytope. This allows us to use LP decoding, with all its theoretical guarantees, to decode large-scale error correcting codes efficiently.

1 Introduction

Feldman *et al.* in [4] proposed a novel decoding algorithm for low density parity check (LDPC) codes based on linear programming (LP). They thereby pose the decoding problem in the framework of convex optimization. They show that the bit-error-rate performance of this LP-based decoder is comparable to iterative decoding and can correct a constant number of errors. The performance of this decoding algorithm can accurately be established using pseudo-codewords. Upon success the algorithm provides a certificate of correctness (ML certificate). Standard LP solvers can be employed to solve the underlying decoding program in poly-time but do not immediately have a distributed nature. In contrast, decoding algorithms that follow the belief propagation framework do not have strong theoretical convergence guarantees but do posses a distributed nature. This is a desirable trait as it leads to scalability via parallel implementations. In this work we exploit the rich structure inherent to the LP formulation to develop efficient distributed algorithms for solving the decoding problem. The result is a theoretically strong method for efficiently decoding modern error correcting codes with large block lengths.

Over the years there has been significant work on decomposition methods and scalable algorithms in the optimization community. One established technique is the *alternating direction method of multipliers* (ADMM) (see Boyd *et al.* [1] and the references therein). ADMM is well suited for distributed convex optimization. Strong convergence guarantees have been established for ADMM. ADMM is a robust method that has successfully been applied to a number of large-scale problems in machine learning and statistics. In this paper we apply ADMM to develop an efficient

*Department of Computer Sciences, University of Wisconsin - Madison. sid@cs.wisc.edu.

†Department of Electrical and Computer Engineering, University of Wisconsin - Madison. xliu94@wisc.edu.

‡Department of Electrical and Computer Engineering, University of Wisconsin - Madison. sdraper@ece.wisc.edu.

§Department of Computer Sciences, University of Wisconsin - Madison. brecht@cs.wisc.edu.

decoupled decoding algorithm. The feasible region of the decoding LP is stated using the so-called parity polytope. In the ADMM framework projecting onto the parity polytope is a non-trivial step. We develop a new characterization of the parity polytope and based on that develop a fast – nearly linear time – projection algorithm, which allows us to solve the decoding LP via ADMM.

2 Background and Related Work

In this paper we consider a binary linear LDPC code $\mathcal{C}$ of length N defined by a $M \times N$ parity-check matrix H. Each of M parity checks, indexed by $\mathcal{J} = \{1, 2, \ldots, M\}$, corresponds to a row in the parity check matrix H. Codeword symbols are indexed by the set $\mathcal{I} = \{1, 2, \ldots, N\}$. The neighborhood of a check j, denoted by $\mathcal{N}_c(j)$, is the set of indices $i \in \mathcal{I}$ that participate in the jth parity check, i.e., $\mathcal{N}_c(j) = \{i \mid H_{j,i} = 1\}$. Similarly for a component $i \in \mathcal{I}$, $\mathcal{N}_v(i) = \{j \mid H_{j,i} = 1\}$. Given a vector $x \in \{0,1\}^N$, the jth parity-check is said to be satisfied if $\sum_{i \in \mathcal{N}_c(j)} x_i$ is even. In other words, the bits assigned to x_i for $i \in \mathcal{N}_c(j)$ have even parity. We say that a length-n binary vector x is a codeword, $x \in \mathcal{C}$, if and only if (iff) all parity checks are satisfied. In a regular LDPC code there is a fixed constant d, such that for all checks $j \in \mathcal{J}$, $|\mathcal{N}_c(j)| = d$. Also for all components $i \in \mathcal{I}$, $|\mathcal{N}_v(i)|$ is a fixed constant. We focus on regular LDPC codes but our techniques and results extend to general LDPC codes. Let P_j be the binary $d \times N$ matrix that selects out the d components of x that participate in the jth check. For example, say the neighborhood of the jth check, $\mathcal{N}_c(j) = \{i_1, i_2, \ldots i_d\}$, where $i_1 < i_2 < \ldots < i_d$. Then, for all $k \in [d]$ the (k, i_k)th entry of P_j is one, the remaining entries are zero. For any codeword $x \in \mathcal{C}$, $P_j x$ is an even parity vector of dimension d for all j.

We begin by describing maximum likelihood decoding and the LP relaxation proposed by Feldman *et al.* Say vector $\tilde{x}$ is received over a binary symmetric channel (BSC), with cross over probability p. Maximum likelihood (ML) decoding selects a codeword $x \in \mathcal{C}$ that maximizes $p(\tilde{x} \mid x)$, the probability that $\tilde{x}$ was received given that x was sent. For the channel at hand we have $p(\tilde{x} \mid x) = \prod_{i \in \mathcal{I}} p(\tilde{x}_i \mid x_i)$. Equivalently, we select a codeword that maximizes $\sum_{i \in \mathcal{I}} \log p(\tilde{x}_i \mid x_i)$. Let γ_i be the negative log-likelihood ratio, $\gamma_i := \log\left(\frac{p(\tilde{x}_i|0)}{p(\tilde{x}_i|1)}\right)$. Then, $\gamma_i = \log\left(\frac{p}{1-p}\right)$ if $\tilde{x}_i = 1$ and $\gamma_i = \log\left(\frac{1-p}{p}\right)$ if $\tilde{x}_i = 0$. Since $\log p(\tilde{x}_i \mid x_i) = -\gamma_i x_i + \log p(\tilde{x}_i \mid 0)$, ML decoding reduces to determining an $x \in \mathcal{C}$ that minimizes $\sum_i \gamma_i x_i$. Thus, ML decoding requires minimizing a linear function over the set of codewords.

The feasible region in the decoding LP is described using the parity polytope, $\mathbb{PP}_d = \text{conv}(\{e \in \{0,1\}^d \mid \|e\|_1 \text{ is even}\})$, the convex hull of all d-dimensional binary vectors with an even number of 1s. Note that for all j, $P_j x$ is a vertex of $\mathbb{PP}_d$. The relaxation proposed by Feldman *et al.* enforces that for all checks $j \in \mathcal{J}$, $P_j x \in \mathbb{PP}_d$ instead of being a vertex. Putting these ingredients together yields the LP:

$$\text{minimize } \gamma^T x \qquad \text{s.t. } P_j x \in \mathbb{PP}_d \;\; \forall j \in \mathcal{J} \tag{1}$$

The underlying structure of the formulation was used by Vontobel and Koetter in [10] to develop distributed message-passing type algorithms to solve the decoding LP. Their algorithms are based on the coordinate-ascent method which, when matched with the appropriate scheduling determined by Burshtein in [2], converge to the optimal solution. In Yedida *et al* [12] "difference-map BP" is developed, a simple distributed algorithm which seems to recover the performance of LP decoding, but dose not have convergence guarantees.

In this paper we frame the LP decoding problem in the template of an ADMM problem. ADMM is distributed, has strong convergence guarantees and, in general, is more robust than coordinate ascent. We give the general formulation of ADMM problems and specialize it to the LP decoding problem in Sec. 3. When developing the ADMM update steps we find that one of the steps require projecting onto the parity polytope. Thus, in Sec. 4 we develop the efficient projection algorithm required. We present numerical results in Sec. 5.

3 Decoupled relaxation and optimization algorithms

In this section we present the ADMM formulation of the LP decoding problem and summarize our contributions. In Sec, 3.1 we introduce the general ADMM template. We specialize the template to our problem in Sec. 3.2. We state the algorithm in Sec. 3.3.

3.1 ADMM formulation

To make the LP (1) fit into the ADMM template we relax x to lie in the hypercube, $x \in [0,1]^N$, and add the auxiliary *"replica"* variables $z_j \in \mathbb{R}^d$ for all $j \in \mathcal{J}$. We work with the following parameterization of the decoding LP.

$$\begin{aligned} \text{minimize} \quad & \gamma^T x \\ \text{subject to} \quad & P_j x = z_j \quad \forall j \in \mathcal{J} \\ & z_j \in \mathbb{PP}_d \quad \forall j \in \mathcal{J} \\ & x \in [0,1]^N \end{aligned} \tag{2}$$

The alternating direction method of multiplies works with an augmented Lagrangian which, for this problem, is

$$L_\mu(x, z, \lambda) := \gamma^T x + \sum_{j \in \mathcal{J}} \lambda_j^T (P_j x - z_j) + \frac{\mu}{2} \sum_{j \in \mathcal{J}} \|P_j x - z_j\|_2^2.$$

Here $\lambda_j \in \mathbb{R}^d$ for $j \in \mathcal{J}$ are the Lagrange multipliers and $\mu > 0$ is a fixed penalty parameter. We use λ and z to succinctly represent the collection of λ_js and z_js respectively. Note that the augmented Lagrangian is obtained by adding the two norm term of the residual to the Lagrangian. Say $\mathcal{X}$ and $\mathcal{Z}$ are the feasible regions for variables x and z respectively (induced by $[0,1]^N$ and the $\mathbb{PP}_d$), ADMM consists of the following iterations:

$$\begin{aligned} x^{k+1} &:= \text{argmin}_{x \in \mathcal{X}} \, L_\mu(x, z^k, \lambda^k) \\ z^{k+1} &:= \text{argmin}_{z \in \mathcal{Z}} \, L_\mu(x^{k+1}, z, \lambda^k) \\ \lambda_j^{k+1} &:= \lambda_j^k + \mu \left(P_j x^{k+1} - z_j^{k+1} \right) \end{aligned}$$

The ADMM update steps involve fixing one variable and minimizing the other. In particular, x^k and z^k are the kth iterate and the updates to the x and z variable are performed in an alternating fashion. We use this framework to solve the LP relaxation proposed by Feldman *et al.* and hence develop a distributed decoding algorithm.

3.2 ADMM Update Steps

The x-update corresponds to fixing z and λ (obtained from the previous iteration or initialized at the beginning) and minimizing $L_\mu(x, z, \lambda)$ subject to $x \in [0,1]^N$. For the augmented Lagrangian at hand the x-update simplifies to

$$x = \Pi_{[0,1]^N}\left(P^{-1} \times \left(\sum_j P_j^T\left(z_j - \frac{1}{\mu}\lambda_j\right) - \frac{1}{\mu}\gamma\right)\right).$$

Here $P = \sum_j P_j^T P_j$ and $\Pi_{[0,1]^N}(\cdot)$ corresponds to projecting onto the hypercube $[0,1]^N$. The latter can easily be accomplished by independently projecting the components onto $[0,1]$. Note that for any j, $P_j^T P_j$ is a $N \times N$ diagonal binary matrix with non-zero entries at (i,i) iff $i \in \mathcal{N}_c(j)$. This implies that $\sum_j P_j^T P_j$ is a diagonal matrix with the (i,i)th entry equal to $|\mathcal{N}_v(i)|$. Hence $P^{-1} = (\sum_j P_j^T P_j)^{-1}$ is a diagonal matrix with $1/|\mathcal{N}_v(i)|$ as the ith diagonal entry.

Component-wise, the update rule corresponds to taking the average of the corresponding replica values, z_j, adjusted by the the scaled dual variable, λ_j/μ, and taking a step in the negative log-likelihood direction. For any $j \in \mathcal{N}_v(i)$ let $z_j^{(i)}$ denote the component of z_j that corresponds to the ith component of x, in other words the ith component of $P_j^T z_j$. Similarly let $\lambda_j^{(i)}$ be the ith component of $P_j^T \lambda_j$. With this notation the update rule for the ith component of x is

$$x_i = \Pi_{[0,1]}\left(\frac{1}{|\mathcal{N}_v(i)|}\left(\sum_{j \in N_v(i)}\left(z_j^{(i)} - \frac{1}{\mu}\lambda_j^{(i)}\right) - \frac{1}{\mu}\gamma_i\right)\right).$$

Component-wise the x-update reduces to a type of averaging, each of which can be done in parallel.

The z-update corresponds to fixing x and λ and minimizing $L_\mu(x, \lambda, z)$ subject to $z_j \in \mathbb{PP}_d$ for all $j \in \mathcal{J}$. The relevant observation here is that the augmented Lagrangian is separable with respect to z_js and hence the minimization step splits into $|\mathcal{J}|$ separate problems that can each be solved independently. This decouples the underlying algorithm making it scalable.

For each $j \in \mathcal{J}$ the update is to find the z_j that minimizes

$$\frac{\mu}{2}\|P_j x - z_j\|_2^2 - \lambda_j^T z_j \quad \text{s.t.} \quad z_j \in \mathbb{PP}_d.$$

Since the values of x and λ are fixed so are $P_j x$ and λ_j/μ. Setting $v = P_j x + \lambda_j/\mu$ and completing the square we get that the desired update z_j^* is

$$z_j^* = \text{argmin}_{\tilde{z} \in \mathbb{PP}_d} \|v - \tilde{z}\|_2^2.$$

Thus, the z-update corresponds to projecting onto the parity polytope.

Recall that the parity polytope $\mathbb{PP}_d$ is the the convex hull of all d-dimensional binary vectors with even Hamming weight. In [6] Jeroslow gives an explicit representation of the parity polytope. Later, in [11] Yannakakis improves this to provide a quadratic (in terms of the dimension d) representation, i.e., the total number of involved constraints in the decoding LP is quadratic in d (see [4]). While this LP can be solved with standard solvers in polynomial time, the quadratic size of the LP might be prohibitive in real-time or embedded decoding applications.

In Sec. 4 we develop a new characterization of the parity polytope. We show that for all vectors $u \in \mathbb{PP}_d$ there exists an even integer $r < d$ such that u can be expressed as a convex combination of d-dimensional binary vectors of Hamming weight r or $r+2$. Of course, any vector in the parity polytope is a convex combination of binary vectors of even Hamming weight. Our characterization shows that, in fact, vectors of only two weights are required. We term the lower weight, r, the "constituent" parity of the vector. The constituent parity is trivially solved for. Based on this representation we develop a near-linear time projection algorithm.

Roughly, our approach is as follows. Given a vector $v \in \mathbb{R}^d$ we first compute r, the constituent parity of its projection. Let $\mathbb{PP}_d^r$ ($\mathbb{PP}_d^{r+2}$) denote the convex hull of all d-dimensional binary vectors of Hamming weight r ($r+2$). Given our characterization, projecting onto the polytope is equivalent to determining an $\alpha \in [0,1]$, a vector $a \in \mathbb{PP}_d^r$, and a vector $b \in \mathbb{PP}_d^{r+2}$ such that the ℓ_2 norm of $v - \alpha a - (1-\alpha)b$ is minimized. We develop an algorithm in Sec. 4.4) that, for a fixed α, determines the optimal $a \in \mathbb{PP}_d^r$ and $b \in \mathbb{PP}_d^{r+2}$. The function $\min_{a \in \mathbb{PP}_d^r,\, b \in \mathbb{PP}_d^{r+2}} \|v - \alpha a - (1-\alpha)b\|_2^2$ is convex in α. Hence we can perform perform a one-dimensional line search (using, for example, the secant method) to determine the optimal value for α and thence the desired projection. The algorithm that, for a given α, solves for the optimal a and b, first projects the given vector onto $\alpha\mathbb{PP}_d^r$ and then projects the residual onto $(1-\alpha)\mathbb{PP}_d^{r+2}$; $\alpha\mathbb{PP}_d^r$ is a scaled version of $\mathbb{PP}_d^r$. Projection onto $\alpha\mathbb{PP}_d^r$ (cf. Sec. 4.4) can be performed in $O(d \log d)$ time using a type of reverse waterfilling algorithm. Thus, our approach gives an efficient method for projecting onto the parity polytope.

3.3 ADMM Decoding Algorithm

The complete ADMM-based algorithm is specified below. We declare convergence when the replicas differ form the optimal x variable by less than some tolerance $\epsilon > 0$.

Algorithm 1 Given a binary N-dimensional vector $\tilde{x} \in \{0,1\}^N$, parity check matrix H, and parameters μ and ϵ, solve the decoding LP specified in (2)

Construct the negative log-likelihood vector γ based on received word $\tilde{x}$.
Construct the $d \times N$ matrix P_j for all $j \in \mathcal{J}$.
Initialize z_j and λ_j as the all zeros vector for all $j \in \mathcal{J}$.
repeat
  Update $x_i \leftarrow \prod_{[0,1]}\Big(\frac{1}{|\mathcal{N}_v(i)|}\Big(\sum_{j \in N_v(i)}\Big(z_j^{(i)} - \frac{1}{\mu}\lambda_j^{(i)}\Big) - \frac{1}{\mu}\gamma_i\Big)\Big)$ for all $i \in \mathcal{I}$.
  for all $j \in \mathcal{J}$ **do**
    Set $v_j = P_j x + \lambda_j/\mu$.
    Update $z_j \leftarrow \Pi_{\mathbb{PP}_d}(v_j)$ where $\Pi_{\mathbb{PP}_d}(\cdot)$ means project onto the parity polytope.
    Update $\lambda_j \leftarrow \lambda_j + \mu\,(P_j x - z_j)$.
  end for
until $\max_j \|P_j x - z_j\|_\infty < \epsilon$ **return** x.

4 Projecting onto the Parity Polytope

In this section we develop our efficient projection algorithm. We set notation in Sec. 4.1. We develop our "two-slice" representation of any point in $\mathbb{PP}_d$ in Sec. 4.2. Given any $u \in \mathbb{R}^d$, in Sec. 4.3 we connect the weight of the projection of u onto $\mathbb{PP}_d$ to the (easily computed) constituent

parity of the projection of u onto the unit hypercube. Finally, in Sec. 4.4 we develop the projection algorithm.

4.1 Notation

Let $\mathbb{P}_d = \left\{e \in \{0,1\}^d \mid \|e\|_1 \text{ is even}\right\}$. The parity polytope $\mathbb{PP}_d = \text{conv}(\mathbb{P}_d)$, the convex hull of $\mathbb{P}_d$. Note that $v \in \mathbb{PP}_d$ iff there exist $e_i \in \mathbb{P}_d$ such that $v = \sum_i \alpha_i e_i$ where $\sum_i \alpha_i = 1$ and $\alpha_i \geq 0$. We denote the set of d-dimensional binary vectors with parity r as $\mathbb{P}_d^r = \{e \in \{0,1\}^d \mid \|e\|_1 = r\}$ and define $\mathbb{PP}_d^r = \text{conv}(\mathbb{P}_d^r)$. Given $a \in \mathbb{R}_+$ by $\lceil a \rceil_{\text{even}}$ we denote the smallest even integer greater than or equal to a and by $\lfloor a \rfloor_{\text{even}}$ the largest even integer less than or equal to a. We denote the projection of a vector v onto a convex set Ω by $\Pi_\Omega(v)$.

4.2 Structural Characterization

Let v and w be d-vectors sorted in decreasing order. The vector w is said to majorize v if

$$\sum_{k=1}^{q} v_k \leq \sum_{k=1}^{q} w_k \quad \forall\ 1 \leq q < d,$$
$$\sum_{k=1}^{d} v_k = \sum_{k=1}^{d} w_k\,.$$

We make use of the following Theorem (see [7] and references therein).

Theorem 1. *Suppose v and w are d-vectors sorted in decreasing order. Then v is in the convex hull of all permutations of w if and only if w majorizes v.*

From this theorem we conclude that a sorted vector $v \in [0,1]^d$ is in the $\mathbb{PP}_d^s$ iff

$$\sum_{k=1}^{q} v_k \leq \min(q, s) \quad \forall\ 1 \leq q < d, \tag{3}$$
$$\sum_{k=1}^{d} v_k = s. \tag{4}$$

The relations follow by noting the first sum is less than q and the second sum must equal s for v to be in $\mathbb{PP}_d^s$.

By definition, any $v \in \mathbb{PP}_d$ can be expressed as a convex combination of the vertices of $\mathbb{PP}_d$. Using $e_i^{(s)}$ to denote the ith vertex of parity s and $\gamma_i^{(s)}$ the corresponding non-negative weighting, we write $v = \sum_{s\ \text{even}}^{d} \sum_i \gamma_i^{(s)} e_i^{(s)}$. Defining $\mu_s = \sum_i \gamma_i^{(s)}$ we have

$$\sum_{s\ \text{even}} \mu_s = 1, \quad \mu \geq 0. \tag{5}$$

Summing the first q coordinates of v and defining $e_{i,k}^{(s)}$ to be the kth component of $e_i^{(s)}$, we apply (3)

to get

$$\sum_{k=1}^{q} v_k = \sum_{s \text{ even}}^{d} \mu_s \sum_{k=1}^{q} \sum_{i} \frac{\gamma_i^{(s)}}{\mu_s} e_{i,k}^{(s)}$$
$$\leq \sum_{s \text{ even}}^{d} \mu_s \min(q, s) \quad \forall\ 1 \leq q < d. \tag{6}$$

In addition, since the $e_i^{(s)}$ are all of weight s, when we sum over all coordinates we get

$$\sum_{k=1}^{d} v_k = \sum_{s \text{ even}}^{d} \mu_s s. \tag{7}$$

Lemma 1. *("Two-slice" lemma) Suppose $v \in \mathbb{PP}_d$ and r is the even integer satisfying $r \leq \sum_{k=1}^{d} v_k \leq r+2$, then v can be expressed as a convex combination of vectors of parity r and $r+2$.*

Let $\alpha, 0 \leq \alpha \leq 1$ be such that $\|v\|_1 = \alpha r + (1-\alpha)(r+2) = r + 2(1-\alpha)$. In the following proof α plays the role of μ_r and $(1-\alpha)$ plays the role of μ_{r+2}. To show Lemma 1 what we need to show is that there is a representation of any $v \in \mathbb{PP}_d$ such that $\mu_s = 0$ for all s except r and $r+2$.

Proof. Given the definition of α and the identifications $\mu_r = \alpha$ and $\mu_{r+2} = (1-\alpha)$, (6) and (7) simplify to

$$\sum_{k=1}^{q} v_k \leq \alpha \min(q, r) + (1-\alpha) \min(q, r+2)$$
$$\forall\ 1 \leq q < d, \tag{8}$$

$$\sum_{k=1}^{d} v_k = \alpha r + (1-\alpha)(r+2). \tag{9}$$

By the definition of α, (9) is satisfied. In (8) the cases $q \leq r$ and $q \geq r+2$ are rather straightforward. For any $q < r$, since there are only q terms in (8) and $v_k \leq 1$ for all k then, e.g., $\min\{q, r\} = q$ and (8) must hold. For any $q \geq r+2$ we use (9) to write $\sum_{k=1}^{q} v_k = \alpha r + (1-\alpha)(r+2) - \sum_{q+1}^{d} v_k \leq \alpha r + (1-\alpha)(r+2)$ since $v_k \geq 0$.

So to prove containment in $\mathbb{PP}_d$, it remains to verify only one more inequality in (8). Namely, we need to show that

$$\sum_{k=1}^{r+1} v_k \leq \alpha r + (1-\alpha)(r+1) = r + (1-\alpha).$$

By assumption, v and μ satisfy (5) and (6). Thus

$$\sum_{k=1}^{r+1} v_k \leq \sum_{s \text{ even}} \mu_s \min(s, r+1) \tag{10}$$

must hold. We now show that the largest value attainable for the right hand side is precisely $r + 1 - \alpha$. To see this, consider the linear program

$$\begin{array}{ll} \text{maximize} & \sum_{s \text{ even}} \mu_s \min(s, r+1) \\ \text{subject to} & \sum_{s \text{ even}} \mu_s s = r + 2(1-\alpha) \\ & \sum_{s \text{ even}} \mu_s = 1 \\ & \mu_s \geq 0. \end{array}$$

The dual program is

$$\begin{array}{ll} \text{minimize} & (r+2-2\alpha)u_1 + u_2 \\ \text{subject to} & u_1 s + u_2 \geq \min(s, r+1) \ \forall \ s \text{ even}. \end{array}$$

Setting $\mu_r = \alpha$, $\mu_{r+2} = (1-\alpha)$, $u_1 = 1/2$, $u_2 = r/2$, and all other primal variables to zero satisfies the Karush-Kuhn-Tucker (KKT) conditions for this primal/dual pair of LPs. The associated optimal cost is $r + 1 - \alpha$. Thus, the right hand side of (10) is at least $r + 1 - \alpha$, completing the proof.

□

Another useful consequence of Theorem 1 is the following corollary.

Corollary 1. *Let v be a vector in $[0,1]^d$. If $\sum_{i=1}^d v_i$ is an even integer then $v \in \mathbb{PP}_d$.*

Proof. Let $\sum_i v_i = s$. Since v is majorized by a sorted binary vector of parity s then, by Theorem 1, $v \in \mathbb{PP}_d^s$ which, in turn, implies $v \in \mathbb{PP}_d$. □

We call the even integer $\lfloor \|v\|_1 \rfloor_{\text{even}}$ the *constituent parity* of vector v.

4.3 Constituent Parity of the Projection

In this section we prove a useful bound on the ℓ_1 norm of the projection of any $u \in \mathbb{R}^d$. This will provides us the constituent parity of the projection.

Lemma 2. *For any vector $u \in \mathbb{R}^d$, denote by ω the projection of u onto $[0,1]^d$ and denote by π the projection of u onto the parity polytope. The following bound holds:*

$$\lfloor \|\omega\|_1 \rfloor_{even} \leq \|\pi\|_1 \leq \lceil \|\omega\|_1 \rceil_{even}.$$

Proof. Let $\rho_U = \lceil \|\omega\|_1 \rceil_{\text{even}}$ and $\rho_L = \lfloor \|\omega\|_1 \rfloor_{\text{even}}$. We prove the following fact: given any $y' \in \mathbb{PP}_d$ with $\|y'\|_1 > \rho_U$ there exits a vector $y \in [0,1]^d$ such that $\|y\|_1 = \rho_U$, $y \in \mathbb{PP}_d$, and $\|u - y\|_2^2 < \|u - y'\|_2^2$. The implication of this fact will be that any vector in the parity polytope with ℓ_1 norm strictly greater that ρ_U cannot be the projection of u. Similarly we can also show that any vector with ℓ_1 norm strictly less than ρ_L cannot be the projection on the parity polytope.

First we construct the vector y based on y' and w. Define the set of "high" values to be the coordinates on which y'_i is greater than w_i, i.e., $\mathcal{H} := \{i \in [d] \mid y'_i > \omega_i\}$. Since by assumption $\|y'\|_1 > \rho_U \geq \|\omega\|_1$ we know that $|\mathcal{H}| \geq 1$. Consider the test vector t defined component-wise as

$$t_i = \begin{cases} \omega_i \text{ if } i \in \mathcal{H} \\ y'_i \text{ else} \end{cases}$$

Note that $\|t\|_1 \leq \|\omega\|_1 \leq \rho_U < \|y'\|_1$. The vector t differs from y' only in $\mathcal{H}$, thus by changing (reducing) components of y' in the set $\mathcal{H}$ we can obtain a vector y such that $\|y\|_1 = \rho_U$. In particular there exists a vector y with $\|y\|_1 = \rho_U$ such that for $i \in \mathcal{H} : y'_i \geq y_i \geq \omega_i$ and for $i \notin \mathcal{H} : y_i = y'_i$. Since the ℓ_1 norm of y is even and is in $[0,1]^d$ we have by Corollary 1 that $y \in \mathbb{PP}_d$.

We next show that for all $i \in \mathcal{H}$, $|u_i - y_i| \leq |u_i - y'_i|$. The inequality will be strict for at least one i yielding $\|u - y\|_2^2 < \|u - y'\|_2^2$ and thereby proving the claim.

We start by noting that $y' \in \mathbb{PP}_d$ so $y'_i \in [0,1]$ for all i. Hence, if for some i, $\omega_i < y'_i$ we must also have $\omega_i < 1$, in which case $u_i \leq \omega_i$ since ω_i is the projection of u_i onto $[0,1]$. In summary, $\omega_i < 1$ iff $u_i < 1$ and when $\omega_i < 1$ then $u_i \leq \omega_i$. Combining with the fact that $y'_i \in [0,1]$ we have that if $y'_i > \omega_i$ then $\omega_i \geq u_i$. Thus for all $i \in \mathcal{H}$ we get $y'_i \geq y_i \geq \omega_i \geq u_i$ where the first inequality is strict for at least one i. Since $y_i = y'_i$ for $i \notin \mathcal{H}$ this means that $|u_i - y_i| \leq |u_i - y'_i|$ for all i where the inequality is strict for at least one value of i. Overall then, $\|u - y\|_2^2 < \|u - y'\|_2^2$ and both $y \in \mathbb{PP}_d$ (by construction) and $y' \in \mathbb{PP}_d$ (by assumption). Thus, y' cannot be the projection of u onto $\mathbb{PP}_d$. Thus the ℓ_1 norm of the projection of u, $\|\pi\|_1 \leq \rho_U$. A similar argument shows that $\|\pi\|_1 \geq \rho_L$ and so $\|\pi\|_1$ must lie in $[\rho_L, \rho_U]$ □

4.4 Projection Algorithm

In this section we develop the projection algorithm. Given a vector $v \in \mathbb{R}^d$ denote by ω the projection of v on $[0,1]^d$ and set $r = \lfloor \|\omega\|_1 \rfloor_{\text{even}}$. From Lemma 2 we know that the constituent parity of the projection of v onto $\mathbb{PP}_d$ is r.

In order to determine the projection using the representation of Sec. 4.2 we need to solve the following quadratic program

$$\begin{aligned} &\min_{\alpha \in [0,1]} \min_{t \in \mathbb{PP}_d^{r+2}} \min_{s \in \mathbb{PP}_d^r} \|v - \alpha s - (1-\alpha)t\|_2^2 \\ &= \min_{\alpha \in [0,1]} \min_{t \in (1-\alpha)\mathbb{PP}_d^{r+2}} \min_{s \in \alpha \mathbb{PP}_d^r} \|v - s - t\|_2^2, \end{aligned} \tag{11}$$

where $\alpha\mathbb{PP}_d^r = \{\alpha y \mid y \in \mathbb{PP}_d^r\}$. In other words $\alpha\mathbb{PP}_d^r$ is the convex hull of the set $\{e \in \{0,\alpha\}^d \mid \|e\|_0 = r\}$.

Projecting onto $\mathbb{PP}_d^r$: To develop our projection onto $\mathbb{PP}_d^r$ we note that, for any $\alpha \in [0,1]$, a vector $y \in \alpha\mathbb{PP}_d^r$ if and only if

$$0 \leq y_i \leq \alpha \;\forall\; i \quad \text{and} \quad \sum_{i=1}^d y_i = \alpha r.$$

The problem of projecting v onto $\alpha\mathbb{PP}_d^r$ is thus:

$$\begin{aligned} \min_y \quad & \frac{1}{2}\|v - y\|_2^2 \\ \text{subject to} \quad & 0 \leq y_i \leq \alpha \;\; \forall\; i \\ & \sum_i y_i = \alpha r. \end{aligned} \tag{12}$$

This problem is equivalent to projecting onto the surface of a ℓ_1 ball of radius αr with box constraints. We develop an algorithm similar to the one in [5] to accomplish this task. The key

difference between projecting onto $\alpha\mathbb{PP}^r_d$ and the problem considered in [5] is that the ℓ_1 norm constraint is enforced as an *inequality* in [5] whereas we wish to impose an *equality*, i.e., we need to determine a vector with ℓ_1 norm *exactly* equal to αr. Recalling that $\Pi_\Omega(v)$ denotes the projection of v onto Ω in the following we use $\Pi_{\alpha\mathbb{PP}^r_d}(\cdot)$ and $\Pi_{(1-\alpha)\mathbb{PP}^{r+2}_d}(\cdot)$ for conciseness.

To develop intuition for the quadratic program (12) we write down the KKT conditions. Since the objective function and the inequality constraints are convex and the equality constraint is affine the KKT conditions are not only necessary but are also sufficient.

We associate dual variables γ_i, μ_i and θ with the constraints and write the corresponding Lagrangian as

$$\mathcal{L}(y,\mu,\gamma,\theta) = \sum_i \frac{1}{2}(v_i - y_i)^2 - \theta\left(r\alpha - \sum_i y_i\right) - \sum_i \mu_i\left(\alpha - y_i\right) - \sum_i \gamma_i y_i.$$

The KKT conditions state that an optimal solution, $y^* = \Pi_{\alpha\mathbb{PP}^r_d}(v)$, satisfies $\nabla\mathcal{L}(y^*_i) = 0$ which, for the above Lagrangian, implies that for all i

$$v_i - y^*_i = \theta^* + \mu^*_i - \gamma^*_i. \tag{13}$$

Furthermore $\sum_i \mu^*_i(\alpha - y^*_i) = 0$ and $\sum_i \gamma^*_i y^*_i = 0$. Hence for all i such that $0 < y^*_i < \alpha$ we must have $\mu^*_i = 0$ and $\gamma^*_i = 0$. In other words, for all i such that $y^*_i \neq 0$ and $y^*_i \neq \alpha$, the difference $v_i - y^*_i$ is exactly the same: $v_i - y^*_i = \theta^*$.

We defer the details of the algorithm along with a proof of correctness and an analysis of its time complexity to the appendix. The algorithm guarantees that the constructed vector satisfies the KKT conditions. The sufficiency of KKT conditions implies that the constructed vector is optimal. The algorithm is a reverse waterfilling type algorithm consisting of two passes.

The algorithm works with three sets: the "clipped" set $\mathcal{C} := \{i \mid y^*_i = \alpha\}$, the "active" set $\mathcal{A} := \{i \mid 0 < y^*_i < \alpha\}$ and the "zero" set $\mathcal{Z} = \{i \mid y^*_i = 0\}$. Using the KKT conditions we now argue that the largest components of v belong to $\mathcal{C}$, the smallest to $\mathcal{Z}$ and the rest to $\mathcal{A}$. First, consider any $i \in \mathcal{C}$ and $j \in \mathcal{A}$. We show that $v_i > v_j$. By the KKT conditions, for any $i \in \mathcal{C}$ we can express v_i as $v_i = \theta^* + \mu^*_i + y^*_i = \theta^* + \mu^*_i + \alpha$ where $\gamma^*_i = 0$. In addition, for any $j \in \mathcal{A}$ we have already noted that $v_j = \theta^* + y^*_j$. Then, since $y^*_j < \alpha$ because $j \in \mathcal{A}$ and since $\mu^*_i \geq 0$, we know that $v_i > v_j$. Next, for any index $k \in \mathcal{Z}$ by the KKT conditions we can write $v_k = \theta^* - \gamma^*_i$ so, since $\gamma^*_i \geq 0$ we see that $v_j > v_k$.

What the above tells us is that we should sort the input vector v component-wise in non-increasing order at the beginning of the algorithm. If we do this the indices in $\mathcal{C}$ will be strictly smaller than the indices in $\mathcal{A}$, and the indices in $\mathcal{A}$ will be strictly smaller that the indices in $\mathcal{Z}$.

However, we still need to determine the change points between sets. To see how to determine these refer to Fig. 1. There is a lower waterfilling level θ^* and an upper level $\theta^* + \alpha$. Components of v larger than the upper level are in $\mathcal{C}$ and components of v smaller than the lower level are in $\mathcal{Z}$. The total waterfilling budget is consumed by the $|\mathcal{C}|$ terms in $\mathcal{C}$ and the contribution of the terms in $\mathcal{A}$. This is equal to the total areas of the shaded bars in the figure at levels θ^* and $\theta^* + \alpha$. Conceptually, our algorithm starts with θ^* high and lowers it until the relation $\alpha|\mathcal{C}| + \sum_{i\in\mathcal{A}}(v_i - \theta^*) = \alpha r$ is satisfied, which follows from the fact that $\sum_i y^*_i = \alpha r$. Numerically we can test all possible sets of break-points. As there are only $2d$ choices this is a linear search. Once the sets have been determined we use the same relation $\alpha|\mathcal{C}| + \sum_{i\in\mathcal{A}}(v_i - \theta^*) = \alpha r$ to solve for θ^*. With the index sets and θ^* at hand we can directly determine individual components of the projection y^*. The algorithmic complexity is dominated by the initial sort, hence it is $O(d \log d)$.

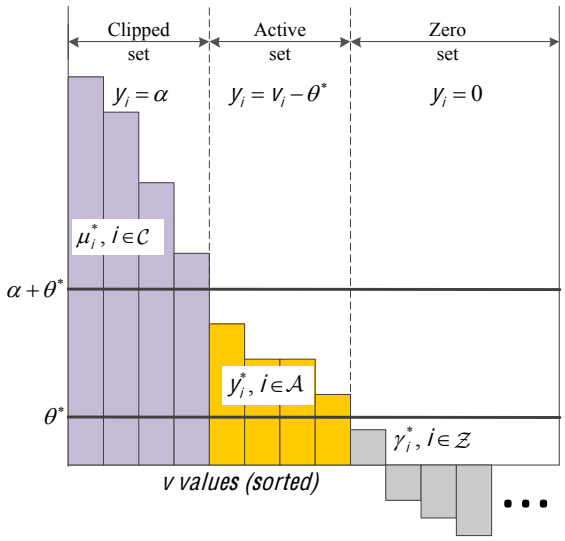

Figure 1: Water-filling Algorithm for determining $\Pi_{\alpha\mathbb{PP}_d^r}(v)$

Projecting onto $\mathbb{PP}_d$: We now develop an efficient algorithm to solve the quadratic program in (11), i.e., to project a vector $v \in \mathbb{R}^d$ onto $\mathbb{PP}_d$. The algorithm executes a binary search over α, for each first projecting the v onto $\alpha\mathbb{PP}_d^r$ and then projecting the residual onto $(1-\alpha)\mathbb{PP}_d^{r+2}$.

Let $\Omega_\alpha = \alpha\mathbb{PP}_d^r + (1-\alpha)\mathbb{PP}_d^{r+2}$, in other words, Ω_α denotes the following convex set: $\{w + w' \mid w \in \alpha\mathbb{PP}_d^r,\, w' \in (1-\alpha)\mathbb{PP}_d^{r+2}\}$. For a given $\alpha \in [0,1]$ let $y_\alpha = \Pi_{\Omega_\alpha}(v)$, i.e., the program stated in (11) with α fixed. We show that Algorithm 2, below, determines y_α. Finally, $\Pi_{\mathbb{PP}_d}(v) = \operatorname{argmin}_{\{y_\alpha\}_{\alpha\in[0,1]}} \|v - y_\alpha\|_2^2$ gives the desired result.

Algorithm 2 Given $v \in \mathbb{R}^d$ and $\alpha \in [0,1]$ determine $\Pi_{\Omega_\alpha}(v)$

1: Set $s' \leftarrow \Pi_{\alpha\mathbb{PP}_d^r}(v)$.
2: Set $t' \leftarrow \Pi_{(1-\alpha)\mathbb{PP}_d^{r+2}}(v - s')$.
3: Return $s' + t'$.

First we make a comment on the computational complexity of the algorithm. Using the water-filling approach the projections in the first two steps of the algorithm can be performed in $O(d \log d)$ time. Since there are only two projections, Algorithm 2 executes in $O(d \log d)$ time.

To prove the correctness of the algorithm we begin by proving a useful fact about t', where t' is determined in the second step of the algorithm.

Lemma 3. *For any vector $t \in (1-\alpha)\mathbb{PP}_d^{r+2}$ we have $(v - s' - t')^T(t - t') \le 0$.*

Proof. Since t' is the projection of $v - s'$ on $(1-\alpha)\mathbb{PP}_d^{r+2}$ we get the desired inequality by the projection theorem. □

Next we show that for any $s \in \alpha\mathbb{PP}_d^r$ we have $(v - s' - t')^T(s - s') \le 0$. Combining this with the previous lemma we get that $(v - s' - t')^T(s + t - s' - t') \le 0$ for any $s \in \alpha\mathbb{PP}_d^r$ and $t \in (1-\alpha)\mathbb{PP}_d^{r+2}$.

Hence for any $u \in \Omega_\alpha$ we have $(v - s' - t')^T(u - s' - t') \leq 0$. And again, by the projection theorem, we get that $s' + t' = \Pi_{\Omega_\alpha}(v)$.

Lemma 4. *For any vector $s \in \alpha\mathbb{PP}^r_d$ we have $(v - s' - t')^T(s - s') \leq 0$.*

Proof. We show that $s' = \Pi_{\alpha\mathbb{PP}^r_d}(v - t')$, that is s' is the projection of $v - t'$ on $\alpha\mathbb{PP}^r_d$, which in turn gives us the required inequality.

When we project v onto $\alpha\mathbb{PP}^r_d$ we get that, for some θ_1, the following conditions hold:

$$s' = \begin{cases} 0 & \text{if } \; v_i \leq \theta_1 \\ \alpha & \text{if } \; v_i - \alpha \geq \theta_1 \\ v_i - \theta_1 & \text{if } \; v_i - \alpha < \theta_1 < v_i \end{cases} . \tag{14}$$

These conditions are derived from the KKT conditions of the corresponding quadratic program as in (13) (cf. Section 4.1 in [5] for a general discussion).

The relations in (14) imply that the index set $\{1, 2, \ldots, d\}$ is partitioned into three parts: $\mathcal{U}$, $\mathcal{M}$ and $\mathcal{L}$. In the first $s'_i = \alpha$, in the second $0 < s'_i < \alpha$, and in the third $s'_i = 0$. This is diagrammed in Fig. 2. Defining $\Delta = v - s'$, the above conditions imply that $\Delta_i \geq \theta_1$ for all $i \in \mathcal{U}$; $\Delta_i = \theta_1$ for all $i \in \mathcal{M}$ and $\Delta_i \leq \theta_1$ for $i \in \mathcal{L}$.

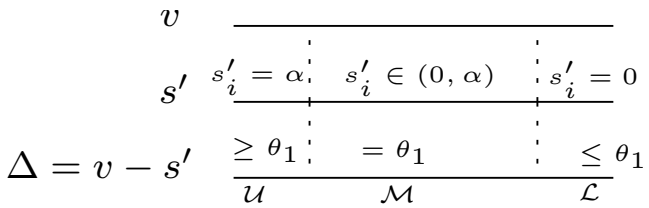

Figure 2: KKT Conditions

Note that $t' = \Pi_{(1-\alpha)\mathbb{PP}^{r+2}_d}(\Delta)$. Defining $\rho = v - s' - t'$, we have that for all indices $i, j \in \mathcal{M}$ the following equality holds: $t'_i = t'_j$. Hence, there exists $\lambda \in \mathbb{R}$ such that for all $i \in \mathcal{M}$ $\rho_i = \lambda$. This follows from writing conditions similar to (14) for t' and input vector Δ. In particular, for some θ_2 we have the following conditions on t'.

$$t' = \begin{cases} 0 & \text{if } \; \Delta_i \leq \theta_2 \\ 1 - \alpha & \text{if } \; \Delta_i - (1 - \alpha) \geq \theta_2 \\ \Delta_i - \theta_2 & \text{if } \; \Delta_i - (1 - \alpha) < \theta_2 < \Delta_i \end{cases} . \tag{15}$$

A useful consequence is that $\rho = \Delta - t'$ is component-wise sorted in the same order as Δ. That is, for $\Delta_1 \geq \Delta_2 \geq \ldots \geq \Delta_d$ we have $\rho_1 \geq \rho_2 \geq \ldots \geq \rho_d$.

Finally we prove that s' satisfies the KKT condition of the following quadratic program with input vector $w = v - t'$. That is s' is the optimal solution of

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w - y\|_2^2 \\ \text{subject to} \quad & 0 \leq y_i \leq \alpha \;\; \forall \; i \\ & \sum_i y_i = \alpha r. \end{aligned}$$

Note that $\rho = w - s'$. For all $i \in \mathcal{M}$ we have by definition that $s'_i \in (0, \alpha)$ and, as stated above, $\rho_i = \lambda$. Since ρ is sorted component-wise we have that for all $i \in \mathcal{U}$, $\rho_i \geq \lambda$. Along similar lines, for all $i \in \mathcal{L}$, $\rho_i \leq \lambda$. Hence s' satisfies the KKT conditions for the above quadratic program, in particular (13) with $\theta^* = \lambda$. Overall this implies that $s' = \Pi_{\alpha \mathbb{PP}_d^r}(w)$. Hence for all $s \in \alpha \mathbb{PP}_d^r$ we have $(w - s')^T(s - s') \leq 0$, which proves the lemma. □

We establish the correctness of Algorithm 2 in the following lemma.

Lemma 5. *Given vector $v \in \mathbb{R}^d$ and scalar $\alpha \in [0, 1]$ let s' and t' be the projections determined by Algorithm 2. Then $s' + t' = \Pi_{\Omega_\alpha}(v)$.*

Proof. By definition for any $u \in \Omega_\alpha$ there exists $s \in \alpha \mathbb{PP}_d^r$ and $t \in (1-\alpha)\mathbb{PP}_d^{r+2}$ such that $u = s + t$. Applying the previous two lemmas we get that $(v - s' - t')^T(s + t - s' - t') \leq 0$ for any $s \in \alpha \mathbb{PP}_d^r$ and $t \in (1 - \alpha)\mathbb{PP}_d^{r+2}$. Therefore for any $u \in \Omega_\alpha$ we have $(v - s' - t')^T(u - s' - t') \leq 0$. Hence by the projection theorem we get that $s' + t' = \Pi_{\Omega_\alpha}(v)$. □

We note that, for a fixed vector v, the function $f(\alpha) = \|v - \Pi_{\Omega_\alpha}(v)\|_2^2$ is convex, and hence we can perform binary search over $\alpha \in [0, 1]$ with Algorithm 2 as a subroutine and in accordance with the level of accuracy determine $\Pi_{\mathbb{PP}_d}(v)$. This implies the following theorem.

Theorem 2. *Given vector $v \in \mathbb{R}^d$ we can determine $\Pi_{\mathbb{PP}_d}(v)$ with $\delta \in [0, 1]$ precision in time $O(d \log d \log(1/\delta))$.*

5 Numerical Results

In this section, we present simulation results for our ADMM decoding algorithm on two different codes. The first code is the (155,64) LDPC code designed by Tanner *et al* [9]. The other code is a (1057,244) LDPC code studied by Yedidia *et al.* in [12].

In Fig. 3 we plot the the error performance of ADMM decoding for the (155,64) code. The parameters used in this simulation are: $\mu = 1.5$, $\epsilon =$ 1e-4, $T_{\max} = 200$ and $\delta =$ 1e-6. For comparison we plot the WER performance of LP decoding using the simplex method, implemented with "adaptive" LP decoding [8], results drawn from [3]. The performance of the two decoders matches closely.

In Fig. 4 we plot results for the (1057,244) code. We again simulated the binary symmetric channel but, to facilitate comparison with [12], we plot the WER performance as a function of the equivalent signal-to-noise ratio (SNR) γ. This is the SNR of a binary-modulated signal transmitted over the additive white Gaussian noise (AWGN) channel coupled with hard-decision decoding. The relationship between crossover probability p and γ is $p = Q(\sqrt{2\gamma})$, where $Q(\cdot)$ is the Q-function. The results from [12] are labeled "E-BP-LP", which is the decoder developed in [12] to get estimates of LP decoding at very low WERs. See [12] for details.

In this set of simulations we investigated the convergence rate of the algorithm by testing performance under different maximum number of iterations: $T_{max} = 50$, $T_{max} = 100$ and $T_{max} = 300$. The other system parameters are $\mu = 2$, $\epsilon =$ 1e-4 and $\delta =$ 1e-6. We also investigated the error performance under different values of the μ parameter. The error performance when $\mu = 10$ and $T_{max} = 300$ is very close to the performance found in [12]. Note also that in both simulations, we accumulate more than 200 decoding errors for each data point.

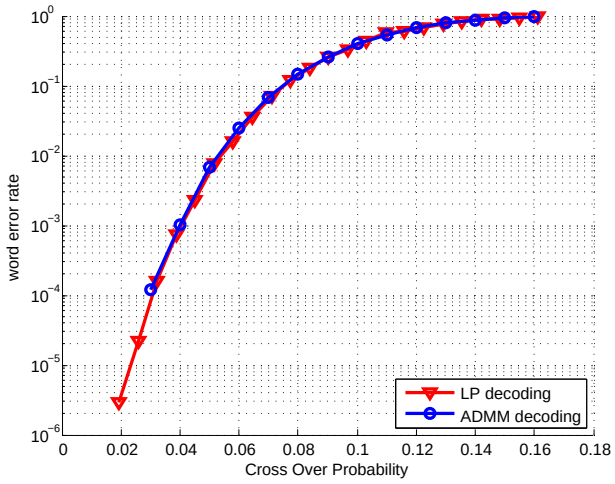

Figure 3: Error performance comparison between ADMM decoding and LP decoding for the (155,64) LDPC code. The WER is plotted as a function of crossover probability.

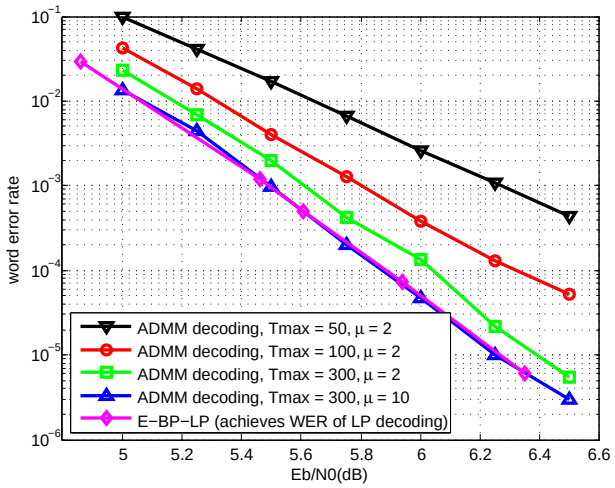

Figure 4: Error performance comparison between ADMM decoding and LP decoding for the (1057,244) LDPC code. The WER is plotted as a function of SNR.

We would like to emphasize two important implementational aspects of our ADMM decoder. First, the one dimensional searching method used to optimize α has a significant impact on the overall efficiency of the decoder. We compared the golden section search with the secant method and results show that the secant method can be three-to-four times faster than the golden section search. This makes the secant method our choice for the simulations. Second, parameters in the algorithm play an important role in the decoder. These parameters include termination precision δ used in the secant method, μ and ϵ in Algorithm 1 and the maximum number of iterations $T_{\max}$ allowed for ADMM. Tuning these parameters can affect both error performance and program efficiency. We have not fully optimized all of these parameters, and our experimental results might be further improved after careful tuning. We defer this tuning and experiments on denser codes for future work.

6 Conclusion

In this paper we apply the ADMM template to the LP decoding problem introduced in [4]. A main technical hurdle was the development of an efficient method of projecting a vector onto the parity polytope. We accomplished this in three steps. We first introduced a new representation of points in the parity polytope. We then used the representation to show that projection can be done in a two-step manner. Finally we showed that each step consists of an efficient waterfilling-type algorithm. We demonstrate the effectiveness of our decoding technique on two codes, on the $(155, 64)$ LDPC code introduced in [9], the second a $(1057, 244)$ LDPC code studied in [12]. In those papers the LP decoding performance of these codes was found using the simplex method as the LP solver. We reproduced those results using our ADMM based method. In contrast to simplex-based methods our ADMM-based method is distributed in nature and scales to larger block lengths.

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. *Machine Learning*, 3(1):1–123, 2010.

[2] D. Burshtein. Iterative approximate linear programming decoding of LDPC codes with linear complexity. *Information Theory, IEEE Transactions on*, 55(11):4835–4859, 2009.

[3] S.C. Draper, J.S. Yedidia, and Y. Wang. ML decoding via mixed-integer adaptive linear programming. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 1656–1660. IEEE, 2007.

[4] J. Feldman, M.J. Wainwright, and D.R. Karger. Using linear programming to decode binary linear codes. *Information Theory, IEEE Transactions on*, 51(3):954–972, 2005.

[5] M.D. Gupta, S. Kumar, and J. Xiao. L_1 projections with box constraints. *Arxiv preprint arXiv:1010.0141*, 2010.

[6] RG Jeroslow. On defining sets of vertices of the hypercube by linear inequalities. *Discrete Mathematics*, 11(2):119–124, 1975.

[7] A.W. Marshall, I. Olkin, and Arnold B.C. *Inequalities: theory of majorization and its applications*. Springer, 2009.

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

[9] R. M. Tanner, D. Sridhara, and T. Fuja. A class of group-structured LDPC codes. In *Proc. ICSTA*, Ambleside, UK, 2001.

[10] P.O. Vontobel and R. Koetter. On low-complexity linear-programming decoding of LDPC codes. *European transactions on telecommunications*, 18(5):509–517, 2007.

[11] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466, 1991.

[12] J.S. Yedidia, Y. Wang, and S.C. Draper. Divide and concur and difference-map BP decoders for LDPC codes. *IEEE Transactions on Information Theory*, 57(2):786–802, 2011.

A Algorithm for projecting onto $\mathbb{PP}^r_d$

In this appendix we show that the water-filling algorithm executes in $O(d \log d)$ time provide a proof of correctness.

Lemma 6. *Given $v \in \mathbb{R}^d$, Algorithm 3 determines $\Pi_{\alpha \mathbb{PP}^r_d}(v)$ in $O(d \log d)$ time.*

Proof. Initially we sort v component-wise. This takes $O(d \log d)$ time. We next iteratively determine the index sets $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{Z}$. Every iteration either increments the cardinality of $\mathcal{C}$ or decrements the cardinality of $\mathcal{Z}$ or both. The total number of iterations can be no more than $2d$. Hence the algorithm executes in time $O(d \log d)$.

To prove the correctness of the algorithm we show that the vector determined by the algorithm satisfies the KKT conditions (13). As stated in the discussion of the projection onto $\mathbb{PP}^r_d$ in Sec. 4.4, if we know the sets $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{Z}$ then computing θ^* amounts to solving the linear equation: $\alpha|\mathcal{C}| + \sum_{i \in \mathcal{A}}(v_i - \theta^*) = \alpha r$. Knowing the sets and θ^* we determine the components of the projection as

$$y^*_i = \begin{cases} \alpha & \text{if} \quad i \in \mathcal{C} \\ v_i - \theta^* & \text{if} \quad i \in \mathcal{A} \\ 0 & \text{if} \quad i \in \mathcal{Z} \end{cases} .$$

This is exactly how the algorithm sets the components, and hence to prove correctness we establish that the algorithm correctly determines the index sets.

The KKT conditions imply that for any $i \in \mathcal{C}$ we have $v_i - \alpha = \theta^* + \mu^*_i$ for $\mu^*_i \geq 0$. Hence $v_i - \alpha$ is a lower bound on θ^*. Also for $k \in \mathcal{Z}$ we have $v_k \leq \theta^*$. We maintain these lower bounds for θ^* as the variable θ in the algorithm. Given candidate sets $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{Z}$ and lower bound θ we compute T, the largest ℓ_1 norm that a vector satisfying these index sets can achieve. If $T < \alpha r$ then we need more components to be non-zero to meet the waterfilling budget with equality and hence we increase the ℓ_1 norm of the tentative projection.

Note that for $i \in \mathcal{C}$ and $k \in \mathcal{Z}$ we must have $v_i - \alpha \geq v_k$. Say at some point c is the highest index of the components in $\mathcal{C}$ and z is the lowest index of the components in $\mathcal{Z}$. Then, the fact that vector v is sorted component-wise implies $\operatorname{argmin}_{i \in \mathcal{C}} v_i = c$ and $\operatorname{argmax}_{k \in \mathcal{Z}} v_k = v_z$. We test whether $v_c - \alpha \geq v_k$ and update the sets so as to maintain the inequality and increase T.

The relevant observation is that if we fix $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{Z}$ and enforce all KKT conditions except for the norm constraint then the ℓ_1 norm of the projection is a linear function of θ. Moreover the ℓ_1 norm increases continuously as $\mathcal{C}$, $\mathcal{A}$ and $\mathcal{Z}$ are updated one index at a time. This implies that once T goes above αr we know that we have overshot, should exit the loop, revert the last update, and compute θ^*. The resulting vector y satisfies the required KKT conditions and hence it is the required projection. □

Algorithm 3 Given $v \in \mathbb{R}^d$ and $\alpha \in [0,1]$ determine its projection on $\alpha \mathbb{PP}_d^r$

Sort components of v in decreasing order, such that $v_1 \geq v_2 \geq \ldots \geq v_d$
Initialize clipped, active and zero index: $c = 0$, $a = 1$ and $z = 2$ {We maintain the invariant that the clipped set $\mathcal{C}$ contains indices 1 through c, initially it is ϕ, active set $\mathcal{A}$ contains $c+1$ to $z-1$ and the zero set $\mathcal{Z}$ contains z to d}
Initialize active sum $S = v_1$ {We enforce $S = \sum_{i \in \mathcal{A}} v_i$ }
Initialize $\theta = v_1$ {We maintain a lower bound for θ^*}
repeat
  if $z \leq d$ **then**
    if $v_{c+1} - \alpha > v_z$ **then**
      Update $c \leftarrow c+1$ and $S \leftarrow S - v_c$ {This corresponds to adding an index to the clipped set $\mathcal{C}$ and hence decrementing the active set}
      Set $\theta = v_c - \alpha$
    else if $v_{c+1} - \alpha < v_z$ **then**
      Update $z \leftarrow z+1$ and $S \leftarrow S + v_{z-1}$ {This corresponds to removing an index from $\mathcal{Z}$}
      Set $\theta = v_{z-1}$
    else
      Update $z \leftarrow z+1$, $c \leftarrow c+1$ and $S \leftarrow S - v_c + v_{z-1}$ {We update both $\mathcal{C}$ and $\mathcal{Z}$}
      Set $\theta = v_{z-1}$
    end if
  else
    Update $c \leftarrow c+1$ and $S \leftarrow S - v_c$
    Set $\theta = v_c - \alpha$
  end if
  Set total sum $T = \alpha c + S - \theta \max\{z - c - 1, 0\}$
until $T < \alpha r$

{We revert the last change to bring down the sum T}
if the last update incremented c **then**
  Update $S \leftarrow S + v_c$ and $c \leftarrow c-1$
else if the last update incremented z **then**
  Update $S \leftarrow S - v_{z-1}$ and $z \leftarrow z-1$
else if both c and z were incremented in the last update **then**
  Update $S \leftarrow S + v_c - v_{z-1}$, $c \leftarrow c-1$ and $z \leftarrow z-1$
end if
if $z > c+1$ **then**
  Set $\theta^* = \frac{\alpha c + S - \alpha r}{z-c-1}$ {With non-empty active set we can compute θ^*}
  Assign components $y_i = \alpha$ for $1 \leq i \leq c$, $y_j = v_j - \theta^*$ for $c < j < z$ and $y_k = 0$ for $z \leq k \leq d$
else if $z = c+1$ **then**
  Assign $y_i = \alpha$ for $1 \leq i \leq c$ and $y_k = 0$ for $z \leq k \leq d$
end if
Return y