Decomposition Methods for Large Scale LP Decoding

Siddharth Barman[*], Xishuo Liu[†], Stark C. Draper[‡], Benjamin Recht [§]

March 2012

Abstract

When binary linear error-correcting codes are used over symmetric channels, a relaxed version of the maximum likelihood decoding problem can be stated as a linear program (LP). This LP decoder can be used to decode at bit-error-rates comparable to state-of-the-art belief propagation (BP) decoders, but with 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, specifically the Alternating Directions Method of Multipliers (ADMM), 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.

We present numerical results for two LDPC codes. The first is the rate-0.5 $[2640, 1320]$ "Margulis" code, the second a rate-0.77 $[1057.244]$ code. The "waterfall" region of LP decoding is seen to initiate at a slightly higher signal-to-noise ratio than for sum-product BP, however an error-floor is not observed for either code, which is not the case for BP. Our implementation of LP decoding using ADMM executes as quickly as our baseline sum-product BP decoder, is fully parallelizable, and can be seen to implement a type of message-passing with a particularly simple schedule.

Keywords. LP Decoding. Alternating Direction Method of Multipliers.

1   Introduction

While the problem of error correction decoding dates back at least to Richard Hamming's seminal work in the 1940s [19], the idea of drawing upon techniques of convex optimization to solve such problems apparently dates only to Jon Feldman's 2003 Ph.D. thesis [13, 15]. Feldman and his collaborators showed that, for binary codes used over symmetric channels, a relaxed version of the maximum likelihood (ML) decoding problem can be stated as a linear program (LP). Considering graph-based low-density parity-check (LDPC) codes, work by Feldman *et al.* and later authors [42] [44] [37] [49] demonstrates that the bit-error-rate performance of LP decoding is competitive with

---

[*]S. Barman is with the Dept. of Computer Sciences, University of Wisconsin, Madison WI 53706 (sid@cs.wisc.edu).

[†]X. Liu is with the Dept. of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706 (xliu94@wisc.edu).

[‡]S. C. Draper is with the Dept. of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706 (sdraper@ece.wisc.edu).

[§]B. Recht is with the Dept. of Computer Sciences, University of Wisconsin, Madison, WI 53706 (brecht@cs.wisc.edu).

that of standard sum-product (and min-sum) belief propagation (BP) decoding. Furthermore, LP decoding comes with a certificate of correctness (ML certificate) [15] – verifying with probability one when the decoder has found the ML codeword. And, if a high-quality expander [10, 14] or high-girth [2] code is used, LP decoding is guaranteed to correct a constant number of bit flips.

A barrier to the adoption of LP decoding is that solving Feldman's relaxation using generic LP algorithms is not computationally competitive with BP. This is because standard LP solvers do not automatically exploit the rich structure inherent to the linear program. Furthermore, unlike BP, standard solvers do not have a distributed nature, limiting their scalability via parallelized (and hardware-compatible) implementation. In this paper we draw upon large-scale decomposition methods from convex optimization to develop an efficient, scalable algorithm for LP decoding. The result is a suite of new techniques for efficient error correction of modern graph-based codes, and insight into the elegant geometry of a fundamental convex object of error-correction, the *parity polytope*.

A real-world motivation for developing efficient LP decoding algorithms comes from applications that have extreme reliability requirements. While suitably designed LDPC codes decoded using BP can achieve near-Shannon performance in the "waterfall" regime where the signal-to-noise ratio (SNR) is close to the code's threshold, they often suffer from an "error floor" in the high SNR regime. This limits the use of LDPCs in applications such as magnetic recording and fiber-optic transport networks. Error floors result from problematic arrangements in the graphical structure of the code (variously termed "pseudocodewords," "near-codewords," "trapping sets," "instantons," "absorbing sets" [18] [22] [27] [33] [11]), from the sub-optimal BP decoding algorithm, and from the particulars of the implementation of BP. Two natural approaches to improving error floor performance are to design codes with fewer problematic arrangements [20] [39] [47] [56] [38] [48]. or to develop improved decoding algorithms. As LP decoders have empirically not yet been observed to suffer from error floors [49, 55], the approach taken herein is the latter.

A second motivation is that an efficient LP decoder can help to develop closer and closer approximations of ML decoders. This is due to the strong theoretical guarantees associated with LP solvers. When the optimum vertex identified by an LP decoder is integer, the ML certificate property ensures that that vertex corresponds to the ML codeword. When the optimum vertex is non-integer (a "pseudocodeword"), one is motivated to tighten the relaxation to eliminate the problematic pseudocodeword, and try again. Various methods for tightening LP relaxations have been proposed [50] [12] [37]. In some settings one can regularly attain ML performance with few additional constraints [55].

In this paper, we produce a fast decomposition algorithm based on the *Alternating Direction Method of Multipliers* [5] (ADMM). This is a classic technique in convex optimization and has gained a good deal of popularity lately for solving problems in compressed sensing [1] and MAP inference in graphical models [29]. As we describe below, when we apply the ADMM algorithm to LP decoding, the algorithm is a message passing algorithm that bears a striking similarity to belief propagation. Variable update their estimates of the true variable assignment based on information (messages) from parity check and measurement nodes. The parity check nodes produce estimated assignments of local variables based on information from the variable nodes.

To an optimization researcher, our application of ADMM would appear quite straight forward. However, our second contribution, beyond a naive implementation of ADMM, is a very efficient computation of the estimates at the parity checks. Each check update requires the computation of a Euclidean projection onto the aforementioned parity polytope. In Section 4, we demonstrate

that this projection can be computed in log-linear time in the degree of the check. This in turn enables us to write LP solvers with wall-clock speeds comparable to (and sometimes much faster) belief propagation decoders.

The structure of the decoding LP has been examined before in pursuit of efficient implementation. The first attempt was by Vontobel and Koetter [43,44] where the authors used the coordinate-ascent method to develop distributed message-passing type algorithms to solve the LP. There is some delicacy in attaining convergence. But, when their approach is matched with an appropriate message-passing schedule, as determined by Burshtein in [6,7], converge to the optimal solution can be attained. Further, interior-point [40] [45] [46] [36] and revised-simplex [24] approaches have also been applied. In a separate approach Yedida *et al* in [55] introduced "Difference-Map BP" decoding which is a simple distributed algorithm that seems to recover the performance of LP decoding, but does not have convergence guarantees.

In this paper we frame the LP decoding problem in the template of ADMM. ADMM is distributed, has strong convergence guarantees, simple scheduling, and, in general, has been observed to be more robust than coordinate ascent. In addition, we do not have to update parameters (step length) between iterations in ADMM. In Section 2 we introduce the LP decoding problem and introduce notation. We set up the general formulation of ADMM problems in Section 3 and specialize the formulation to the LP decoding problem. In Section 4 we present our main technical contributions wherein we develop the efficient projection algorithm required. We present numerical results in Section 5 and make some final remarks in Section 6.

2 Background

In this paper we consider a binary linear LDPC code $\mathcal{C}$ of length N defined by a $M \times N$ parity-check matrix $\boldsymbol{H}$. Each of the M parity checks, indexed by $\mathcal{J} = \{1, 2, \ldots, M\}$, corresponds to a row in the parity check matrix $\boldsymbol{H}$. 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., $\mathcal{N}_c(j) = \{i \mid \boldsymbol{H}_{j,i} = 1\}$. Similarly for a component $i \in \mathcal{I}$, $\mathcal{N}_v(i) = \{j \mid \boldsymbol{H}_{j,i} = 1\}$. Given a vector $\boldsymbol{x} \in \{0, 1\}^N$, 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 set of values assigned to the x_i for $i \in \mathcal{N}_c(j)$ have even parity. We say that a length-n binary vector $\boldsymbol{x}$ is a codeword, $\boldsymbol{x} \in \mathcal{C}$, 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. For simplicity of exposition we focus our discussion on regular LDPC codes. Our techniques and results extend immediately to general LDPC codes and to high density parity check codes as well.

To denote compactly the subset of coordinates of $\boldsymbol{x}$ that participate in the jth check we introduce the matrix $\boldsymbol{P}_j$. The matrix $\boldsymbol{P}_j$ is the binary $d \times N$ matrix that selects out the d components of $\boldsymbol{x}$ 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 $\boldsymbol{P}_j$ is one and the remaining entries are zero. For any codeword $\boldsymbol{x} \in \mathcal{C}$ and for any j, $\boldsymbol{P}_j \boldsymbol{x}$ is an even parity vector of dimension d. In other words we say that $\boldsymbol{P}_j \boldsymbol{x} \in \mathbb{P}_d$ for all $j \in \mathcal{J}$ (a "local codeword" constraint) where $\mathbb{P}_d$ is defined as

$$\mathbb{P}_d = \{\boldsymbol{e} \in \{0, 1\}^d \mid \|\boldsymbol{e}\|_1 \text{ is even}\}. \tag{2.1}$$

Thus, $\mathbb{P}_d$ is the set of codewords (the codebook) of the length-d single parity-check code.

We begin by describing maximum likelihood (ML) decoding and the LP relaxation proposed by Feldman *et al.* Say vector $\tilde{\boldsymbol{x}}$ is received over a discrete memoryless channel described by channel law (conditional probability) $W : \mathcal{X} \times \tilde{\mathcal{X}} \to \mathbb{R}_{\geq 0}$, $\sum_{\tilde{x} \in \tilde{\mathcal{X}}} W(\tilde{x}|x) = 1$ for all $x \in \mathcal{X}$. Since the development is for binary codes $|\mathcal{X}| = 2$. There is no restriction on $\tilde{\mathcal{X}}$. Maximum likelihood decoding selects a codeword $\boldsymbol{x} \in \mathcal{C}$ that maximizes $p_{\tilde{\boldsymbol{X}}|\boldsymbol{X}}(\tilde{\boldsymbol{x}}|\boldsymbol{x})$, the probability that $\tilde{\boldsymbol{x}}$ was received given that $\boldsymbol{x}$ was sent. For discrete memoryless channel W, $p_{\tilde{\boldsymbol{X}}|\boldsymbol{X}}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \prod_{i \in \mathcal{I}} W(\tilde{x}_i|x_i)$. Equivalently, we select a codeword that maximizes $\sum_{i \in \mathcal{I}} \log W(\tilde{x}_i|x_i)$. Let γ_i be the negative log-likelihood ratio, $\gamma_i := \log\left[W(\tilde{x}_i|0)/W(\tilde{x}_i|1)\right]$. Since $\log W(\tilde{x}_i|x_i) = -\gamma_i x_i + \log W(\tilde{x}_i|0)$, ML decoding reduces to determining an $\boldsymbol{x} \in \mathcal{C}$ that minimizes $\boldsymbol{\gamma}^T \boldsymbol{x} = \sum_{i \in \mathcal{I}} \gamma_i x_i$. Thus, ML decoding requires minimizing a linear function over the set of codewords.[1]

Feldman *et al.* [15] show that ML decoding is equivalent to minimizing a linear cost over the convex hull of all codewords. In other words, minimize $\boldsymbol{\gamma}^T \boldsymbol{x}$ subject to $\boldsymbol{x} \in \text{conv}(\mathcal{C})$. The feasible region of this program is termed the "codeword" polytope. However, this polytope cannot be described tractably. Feldman's approach is first to relax each local codeword constraint $\boldsymbol{P}_j \boldsymbol{x} \in \mathbb{P}_d$ to $\boldsymbol{P}_j \boldsymbol{x} \in \mathbb{PP}_d$ where

$$\mathbb{PP}_d = \text{conv}(\mathbb{P}_d) = \text{conv}(\{\boldsymbol{e} \in \{0,1\}^d \mid \|\boldsymbol{e}\|_1 \text{ is even}\}). \tag{2.2}$$

The object $\mathbb{PP}_d$ is called the "parity polytope". It is the codeword polytope of the single parity-check code (of dimension d). Thus, for any codeword $\boldsymbol{x} \in \mathcal{C}$, $\boldsymbol{P}_j \boldsymbol{x}$ is a vertex of $\mathbb{PP}_d$ for all j. When the constraints $\boldsymbol{P}_j \boldsymbol{x} \in \mathbb{PP}_d$ are intersected for all $j \in \mathcal{J}$ the resulting feasible space is termed the "fundamental" polytope. Putting these ingredients together yields the LP relaxation that we study:

$$\text{minimize } \boldsymbol{\gamma}^T \boldsymbol{x} \quad \text{s.t.} \quad \boldsymbol{P}_j \boldsymbol{x} \in \mathbb{PP}_d \ \ \forall \, j \in \mathcal{J}. \tag{2.3}$$

The statement of the optimization problem in (2.3) makes it apparent that compact representation of the parity polytope $\mathbb{PP}_d$ is crucial for efficient solution of the LP. Study of this polytope dates back some decades. In [21] Jeroslow gives an explicit representation of the parity polytope and shows that it has an exponential number of vertices and facets in d. Later, in [51], Yannakakis shows that the parity polytope has *small lift*, meaning that it is the projection of a polynomially faceted polytope in a dimension polynomial in d. Indeed, Yannakakis' representation requires a quadratic number of variables and inequalities. This is one of the descriptions discussed in [15] to state the LP decoding problem.

Yannakakis' representation of a vector $\boldsymbol{u} \in \mathbb{PP}_d$ consists of variables $\mu_s \in [0,1]$ for all even $s \leq d$. Variable μ_s indicates the contribution of binary (zero/one) vectors of Hamming weight s to $\boldsymbol{u}$. Since $\boldsymbol{u}$ is a convex combination of even-weight binary vectors, $\sum_{\text{even } s} \mu_s = 1$. In addition, variables $z_{i,s}$ are used to indicate the contribution to u_i, the ith coordinate of $\boldsymbol{u}$ made by binary vectors of Hamming weight s. Overall, the following set of inequalities over $O(d^2)$ variables characterize the

---

[1]This derivation applies to all binary-input DMCs. In the simulations of Section 5 we focus on the binary-input additive white Gaussian noise (AWGN) channel. To help make the definitions more tangible we now summarize how they specialize for the binary symmetric channel (BSC) with crossover probability p. For the BSC $\tilde{x}_i \in \{0, 1\}$. If $\tilde{x}_i = 1$ then $\gamma_i = \log[W(1|0)/W(1|1)] = \log[p/(1-p)]$ and if $\tilde{x}_i = 0$ then $\gamma_i = \log[W(0|0)/W(0|1)] = \log[(1-p)/p]$.

4

parity polytope (see [51] and [15] for a proof).

$$0 \le u_i \le 1 \qquad \forall \quad i \in [d]$$
$$0 \le z_{i,s} \le \mu_s \quad \forall \quad i \in [d]$$
$$\sum_{\text{even } s}^{d} \mu_s = 1$$
$$u_i = \sum_{\text{even } s}^{d} z_{i,s} \quad \forall \quad i \in [d]$$
$$\sum_{i=1}^{d} z_{i,s} = s\mu_s \quad \forall \quad s \text{ even}, s \le d.$$

This LP can be solved with standard solvers in polynomial time. However, the quadratic size of the LP prohibits its solution with standard solvers in real-time or embedded decoding applications. In Section 4.2 we show that any vector $\boldsymbol{u} \in \mathbb{PP}_d$ can always be expressed as a convex combination of binary vectors of Hamming weight r and $r+2$ for some even integer r. Based on this observation we develop a new formulation for the parity polytope that consists of $O(d)$ variables and constraints. This is a key step towards the development of an efficient decoding algorithm. Its smaller description complexity also makes our formulation particularly well suited for high-density codes whose study we leave for future work.

3 Decoupled relaxation and optimization algorithms

In this section we present the ADMM formulation of the LP decoding problem and summarize our contributions. In Section 3.1 we introduce the general ADMM template. We specialize the template to our problem in Section 3.2. We state the algorithm in Section 3.3 and frame it in the language of message-passing in Section 3.4.

3.1 ADMM formulation

To make the LP (2.3) fit into the ADMM template we relax $\boldsymbol{x}$ to lie in the hypercube, $\boldsymbol{x} \in [0,1]^N$, and add the auxiliary "replica" variables $\boldsymbol{z}_j \in \mathbb{R}^d$ for all $j \in \mathcal{J}$. We work with a decoupled parameterization of the decoding LP.

$$\begin{aligned}
\text{minimize} \quad & \boldsymbol{\gamma}^T \boldsymbol{x} \\
\text{subject to} \quad & \boldsymbol{P}_j \boldsymbol{x} = \boldsymbol{z}_j \quad \forall\, j \in \mathcal{J} \\
& \boldsymbol{z}_j \in \mathbb{PP}_d \quad \forall\, j \in \mathcal{J} \\
& \boldsymbol{x} \in [0,1]^N.
\end{aligned} \qquad (3.1)$$

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

$$L_\mu(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) := \boldsymbol{\gamma}^T \boldsymbol{x} + \sum_{j \in \mathcal{J}} \boldsymbol{\lambda}_j^T (\boldsymbol{P}_j \boldsymbol{x} - \boldsymbol{z}_j)$$
$$+ \frac{\mu}{2} \sum_{j \in \mathcal{J}} \|\boldsymbol{P}_j \boldsymbol{x} - \boldsymbol{z}_j\|_2^2. \qquad (3.2)$$

Here $\boldsymbol{\lambda}_j \in \mathbb{R}^d$ for $j \in \mathcal{J}$ are the Lagrange multipliers and $\mu > 0$ is a fixed penalty parameter. We use $\boldsymbol{\lambda}$ and $\boldsymbol{z}$ to succinctly represent the collection of $\boldsymbol{\lambda}_j$s and $\boldsymbol{z}_j$s respectively. Note that the augmented Lagrangian is obtained by adding the two-norm term of the residual to the ordinary Lagrangian. The Lagrangian without the augmentation can be optimized via a dual subgradient ascent method [4], but our experiments with this approach required far too many message passing iterations for practical implementation. The augmented Lagrangian smoothes the dual problem leading to much faster convergence rates in practice [31]. For the interested reader, we provide a discussion of the standard dual ascent method in the appendix.

Let $\mathcal{X}$ and $\mathcal{Z}$ denote the feasible regions for variables $\boldsymbol{x}$ and $\boldsymbol{z}$ respectively: $\mathcal{X} = [0,1]^N$ and we use $\boldsymbol{z} \in \mathcal{Z}$ to mean that $\boldsymbol{z}_1 \times \boldsymbol{z}_2 \times \ldots \times \boldsymbol{z}_{|\mathcal{J}|} \in \mathbb{PP}_d \times \mathbb{PP}_d \times \ldots \times \mathbb{PP}_d$, the $|\mathcal{J}|$-fold product of $\mathbb{PP}_d$. Then we can succinctly write the iterations of ADMM as

$$\boldsymbol{x}^{k+1} := \operatorname{argmin}_{\boldsymbol{x} \in \mathcal{X}} L_\mu(\boldsymbol{x}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k)$$
$$\boldsymbol{z}^{k+1} := \operatorname{argmin}_{\boldsymbol{z} \in \mathcal{Z}} L_\mu(\boldsymbol{x}^{k+1}, \boldsymbol{z}, \boldsymbol{\lambda}^k)$$
$$\boldsymbol{\lambda}_j^{k+1} := \boldsymbol{\lambda}_j^k + \mu \left(\boldsymbol{P}_j \boldsymbol{x}^{k+1} - \boldsymbol{z}_j^{k+1} \right).$$

The ADMM update steps involve fixing one variable and minimizing the other. In particular, $\boldsymbol{x}^k$ and $\boldsymbol{z}^k$ are the kth iterate and the updates to the $\boldsymbol{x}$ and $\boldsymbol{z}$ 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 $\boldsymbol{x}$-update corresponds to fixing $\boldsymbol{z}$ and $\boldsymbol{\lambda}$ (obtained from the previous iteration or initialization) and minimizing $L_\mu(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda})$ subject to $\boldsymbol{x} \in [0,1]^N$. Taking the gradient of (3.2), setting the result to zero, and limiting the result to the hypercube $\mathcal{X} = [0,1]^N$, the $\boldsymbol{x}$-update simplifies to

$$\boldsymbol{x} = \Pi_{[0,1]^N} \left(\boldsymbol{P}^{-1} \times \left(\sum_j \boldsymbol{P}_j^T \left(\boldsymbol{z}_j - \frac{1}{\mu} \boldsymbol{\lambda}_j \right) - \frac{1}{\mu} \boldsymbol{\gamma} \right) \right),$$

where $\boldsymbol{P} = \sum_j \boldsymbol{P}_j^T \boldsymbol{P}_j$ 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 $[0,1]$: setting the components that are greater than 1 equal to 1, the components less than 0 equal to 0, and leaving the remaining coordinates unchanged. Note that for any j, $\boldsymbol{P}_j^T \boldsymbol{P}_j$ is a $N \times N$ diagonal binary matrix with non-zero entries at (i,i) if and only if i participates in the jth parity check ($i \in \mathcal{N}_c(j)$). This implies that $\sum_j \boldsymbol{P}_j^T \boldsymbol{P}_j$ is a diagonal matrix with the (i,i)th entry equal to $|\mathcal{N}_v(i)|$. Hence $\boldsymbol{P}^{-1} = (\sum_j \boldsymbol{P}_j^T \boldsymbol{P}_j)^{-1}$ 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, $\boldsymbol{z}_j$, adjusted by the the scaled dual variable, $\boldsymbol{\lambda}_j/\mu$, 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 $\boldsymbol{z}_j$ that corresponds to the ith component of $\boldsymbol{x}$, in other words the ith component of $\boldsymbol{P}_j^T \boldsymbol{z}_j$. Similarly let $\lambda_j^{(i)}$ be the ith component of $\boldsymbol{P}_j^T \boldsymbol{\lambda}_j$. With this notation the update rule for the ith component of $\boldsymbol{x}$ is

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

6

Each variable update can be done in parallel.

The z-update corresponds to fixing $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ and minimizing $L_\mu(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{z})$ subject to $\boldsymbol{z}_j \in \mathbb{PP}_d$ for all $j \in \mathcal{J}$. The relevant observation here is that the augmented Lagrangian is separable with respect to the $\boldsymbol{z}_j$s and hence the minimization step can be decomposed (or "factored") into $|\mathcal{J}|$ separate problems, each of which be solved independently. This decouples the overall problem, making the approach scalable.

We start from (3.2) and concentrate on the terms that involve $\boldsymbol{z}_j$. For each $j \in \mathcal{J}$ the update is to find the $\boldsymbol{z}_j$ that minimizes

$$\frac{\mu}{2}\|\boldsymbol{P}_j\boldsymbol{x} - \boldsymbol{z}_j\|_2^2 - \boldsymbol{\lambda}_j^T \boldsymbol{z}_j \quad \text{s.t.} \quad \boldsymbol{z}_j \in \mathbb{PP}_d.$$

Since the values of $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ are fixed, so are $\boldsymbol{P}_j\boldsymbol{x}$ and $\boldsymbol{\lambda}_j/\mu$. Setting $\boldsymbol{v} = \boldsymbol{P}_j\boldsymbol{x} + \boldsymbol{\lambda}_j/\mu$ and completing the square we get that the desired update $\boldsymbol{z}_j^*$ is

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

The z-update thus corresponds to $|\mathcal{J}|$ projections onto the parity polytope.

3.3 ADMM Decoding Algorithm

The complete ADMM-based algorithm is specified in the Algorithm 1 box. We declare convergence when the replicas differ from the $\boldsymbol{x}$ variables by less than some tolerance $\epsilon > 0$.

Algorithm 1 Given a binary N-dimensional vector $\tilde{\boldsymbol{x}} \in \{0,1\}^N$, parity check matrix $\boldsymbol{H}$, and parameters μ and ϵ, solve the decoding LP specified in (3.1)

1: Construct the negative log-likelihood vector $\boldsymbol{\gamma}$ based on received word $\tilde{\boldsymbol{x}}$.
2: Construct the $d \times N$ matrix $\boldsymbol{P}_j$ for all $j \in \mathcal{J}$.
3: Initialize $\boldsymbol{z}_j$ and $\boldsymbol{\lambda}_j$ as the all zeros vector for all $j \in \mathcal{J}$.
4: **repeat**
5:     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 $i \in \mathcal{I}$.
6:     **for all** $j \in \mathcal{J}$ **do**
7:         Set $\boldsymbol{v}_j = \boldsymbol{P}_j\boldsymbol{x} + \boldsymbol{\lambda}_j/\mu$.
8:         Update $\boldsymbol{z}_j \leftarrow \Pi_{\mathbb{PP}_d}(\boldsymbol{v}_j)$ where $\Pi_{\mathbb{PP}_d}(\cdot)$ means project onto the parity polytope.
9:         Update $\boldsymbol{\lambda}_j \leftarrow \boldsymbol{\lambda}_j + \mu(\boldsymbol{P}_j\boldsymbol{x} - \boldsymbol{z}_j)$.
10:    **end for**
11: **until** $\max_j \|\boldsymbol{P}_j\boldsymbol{x} - \boldsymbol{z}_j\|_\infty < \epsilon$ **return** $\boldsymbol{x}$.

3.4 ADMM Decoding as Message Passing Algorithm

We now present a message-passing interpretation of the ADMM decoding algorithm, Algorithm 1.[2] We establish this interpretation using the "normal" factor graph representation [16] (sometimes also called "Forney-style" factor graphs). One key difference between normal factor graphs and ordinary factor graphs is that the variables in normal factor graph representation are associated with the

---

[2]See [53] for another recent interpretation of ADMM as a message-passing algorithm.

edges of a regular factor graphs [23], and the constraints of the normal graph representation are associated with both factor and variable nodes of the regular representation. See [16,25] for details. In representing the ADMM algorithm as a message-passing algorithm the $\boldsymbol{x}$ and the replicas $\boldsymbol{z}$ are the variables in the normal graph.

We denote by $x_{ij}(k)$ the replica associated with the edge joining node $i \in \mathcal{I}$ and node $j \in \mathcal{J}$, where k indicates the kth iteration. Note that $x_{ij_1}(k) = x_{ij_2}(k) = x_i^k$ for all $j_1, j_2 \in \mathcal{J}$, where x_i^k is the value of x_i at kth iteration in Algorithm 1. The "message" $m_{i \to j}(k) := x_{ij}(k)$ is passed from node i to node j at the beginning of the kth iteration. Incoming messages to check node j are denoted as $\boldsymbol{m}_{\to j}(k) := \{m_{i \to j}(k) : i \in \mathcal{N}_c(j)\}$. The $\boldsymbol{z_j}$ can also be interpreted as the messages passed from check node j to the variable nodes in $\mathcal{N}_c(j)$, denoted as $\boldsymbol{m}_{j \to}(k) := \{m_{j \to i}(k) : i \in \mathcal{N}_c(j)\}$. Let $\boldsymbol{\lambda'_j} := \boldsymbol{\lambda_j}/\mu$ and $\lambda'_{j,i} := \lambda_j^{(i)}/\mu$. Then, for all $j \in \mathcal{N}_v(i)$

$$m_{i \to j}(k+1) =$$

$$\Pi_{[0,1]} \left(\frac{1}{|\mathcal{N}_v(i)|} \sum_{j \in \mathcal{N}_c(j)} \left[m_{j \to i}(k) - \lambda'_{j,i}(k) \right] - \frac{\gamma_i}{\mu} \right).$$

The $\boldsymbol{z}$-update can be rewritten as

$$\boldsymbol{m}_{j \to}(k+1) = \Pi_{\mathbb{PP}_d} \left(\boldsymbol{m}_{\to j}(k) + \boldsymbol{\lambda'_j}(k) \right).$$

The λ'_j updated is

$$\boldsymbol{\lambda'_j}(k+1) = \boldsymbol{\lambda'_j}(k) + \left(\boldsymbol{m}_{\to j}(k) - \boldsymbol{m}_{j \to}(k) \right).$$

With this interpretation, it is clear that the ADMM algorithm decouples the decoding problem and can be performed in a distributed manner.

4 The geometric structure of $\mathbb{PP}_d$, and efficient projection onto $\mathbb{PP}_d$

In this section we develop our efficient projection algorithm. Recall that $\mathbb{P}_d = \left\{ \boldsymbol{e} \in \{0,1\}^d \mid \|\boldsymbol{e}\|_1 \text{ is even} \right\}$ and that $\mathbb{PP}_d = \text{conv}(\mathbb{P}_d)$. Generically we say that a point $\boldsymbol{v} \in \mathbb{PP}_d$ if and only if there exist a set of $\boldsymbol{e}_i \in \mathbb{P}_d$ such that $\boldsymbol{v} = \sum_i \alpha_i \boldsymbol{e}_i$ where $\sum_i \alpha_i = 1$ and $\alpha_i \geq 0$. In contrast to this generic representation, the initial objective of this section is to develop a novel "two-slice" representation of any point $\boldsymbol{v} \in \mathbb{PP}_d$: namely that any such vector can be written as a convex combination of vectors with Hamming weight r and $r + 2$ for some even integer r. We will then use this representation to construct an efficient projection.

We open the section in Section 4.1 by describing the structured geometry of $\mathbb{PP}_d$ that we leverage, and laying out the results that will follow in ensuing sections. In Section 4.2, we prove a few necessary lemmas illustrating some of the symmetry structure of the parity polytope. In Section 4.3 we develop the two-slice representation and connect the ℓ_1-norm of the projection of any $\boldsymbol{v} \in \mathbb{R}^d$ onto $\mathbb{PP}_d$ to the (easily computed) "constituent parity" of the projection of $\boldsymbol{v}$ onto the unit hypercube. In Section 4.4 we present the projection algorithm.

4.1 Introduction to the geometry of $\mathbb{PP}_d$

In this section we discuss the geometry of $\mathbb{PP}_d$. We develop intuition and foreshadow the results to come. We start by making a few observations about $\mathbb{PP}_d$.

- First, we can classify the vertices of $\mathbb{PP}_d$ by their weight. We do this by defining $\mathbb{P}_d^r$, the constant-weight analog of $\mathbb{P}_d$, to be the set of weight-r vertices of $\mathbb{PP}_d$:

$$\mathbb{P}_d^r = \{\mathbf{e} \in \{0,1\}^d \mid \|\mathbf{e}\|_1 = r\}, \tag{4.1}$$

  i.e., the constant-weight-r subcode of $\mathbb{P}_d$. Since all elements of $\mathbb{P}_d$ are in some $\mathbb{P}_d^r$ for some even r, $\mathbb{P}_d = \cup_{0 \leq r \leq d : r \text{ even}} \mathbb{P}_d^r$. This gives us a new way to think about characterizing the parity polytope,

$$\mathbb{PP}_d = \text{conv}(\cup_{0 \leq r \leq d : r \text{ even}} \mathbb{P}_d^r).$$

- Second, we define $\mathbb{PP}_d^r$ to be the convex hull of $\mathbb{P}_d^r$,

$$\mathbb{PP}_d^r = \text{conv}(\mathbb{P}_d^r) = \text{conv}(\{\mathbf{e} \in \{0,1\}^d \mid \|\mathbf{e}\|_1 = r\}). \tag{4.2}$$

  This object is a "permutahedron", so termed because it is the convex hull of all permutations of a single vector; in this case a length-d binary vector with r ones. Of course,

$$\mathbb{PP}_d = \text{conv}(\cup_{0 \leq r \leq d : r \text{ even}} \mathbb{PP}_d^r).$$

- Third, define the affine hyper-plane consisting of all vectors whose components sum to r as

$$\mathcal{H}_d^r = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{1}^T \mathbf{x} = r\}$$

  where $\mathbf{1}$ is the length-d all-ones vector. We can visualize $\mathbb{PP}_d^r$ as a "slice" through the the parity polytope defined as the intersection of $\mathcal{H}_d^r$ with $\mathbb{PP}_d$. In other words, a definition of $\mathbb{PP}_d^r$ equivalent to (4.2) is

$$\mathbb{PP}_d^r = \mathbb{PP}_d \cap \mathcal{H}_d^r,$$

  for r an even integer.

- Finally, we note that the $\mathbb{PP}_d^r$ are all parallel. This follows since all vectors lying in any of these permutahedra are orthogonal to $\mathbf{1}$. We can think of the line segment that connects the origin to $\mathbf{1}$ as the major axis of the parity polytope with each "slice" orthogonal to the axis.

The above observations regarding the geometry of $\mathbb{PP}_d$ are illustrated in Fig. 1. Our development will be as follows. First, in Sec. 4.2 we draw on a theorem from [28] about the geometry of permutahedra to assert that a point $\mathbf{v} \in \mathbb{R}^d$ is in $\mathbb{PP}_d^r$ if and only if a sorted version of $\mathbf{v}$ is *majorized* (see Definition 4.3) by the length-d vector consisting of r ones followed by $d - r$ zeros (the sorted version of any vertex of $\mathbb{PP}_d^r$). This allows us to characterize the $\mathbb{PP}_d^r$ easily.

Second, we rewrite any point $\mathbf{u} \in \mathbb{PP}_d$ as, per our second bullet above, a convex combination of points in slices of different weights r. In other words $\mathbf{u} = \sum_{0 \leq r \leq d : r \text{ even}} \alpha_r \mathbf{u}_r$ where $\mathbf{u}_r \in \mathbb{PP}_d^r$ and the α_r are the convex weightings. We develop a useful characterization of $\mathbb{PP}_d$, the "two-slice" Lemma 4.6, that shows that two slices always suffices. In other words we can always write $\mathbf{u} = \alpha \mathbf{u}_r + (1 - \alpha) \mathbf{u}_{r+2}$ where $\mathbf{u}_r \in \mathbb{PP}_d^r$, $\mathbf{u}_{r+2} \in \mathbb{PP}_d^{r+2}$, $0 \leq \alpha \leq 1$, and $r = \lfloor \|\mathbf{u}\| \rfloor_{\text{even}}$, where $\lfloor a \rfloor_{\text{even}}$ is the largest even integer less than or equal to a. We term the lower weight, r, the "constituent" parity of the vector.

Third, in Sec. 4.3 we show that given a point $\mathbf{v} \in \mathbb{R}^d$ that we wish to project onto $\mathbb{PP}_d$, it is easy to identify the constituent parity of the projection. To express this formally, let $\Pi_{\mathbb{PP}_d}(\mathbf{v})$ be
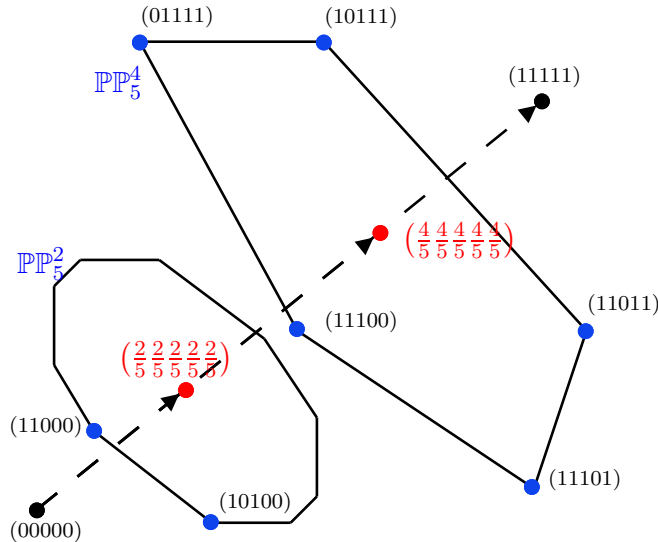
Figure 1: The parity polytope $\mathbb{PP}_d$ can be expressed as the convex hull of "slices" through $\mathbb{PP}_d$, each of which contains all weight-r vertices. These sets, $\mathbb{PP}_d^r$ are permutahedra. They are all orthogonal to the line segment connecting the origin to the all-ones vector. The geometry is sketched for $d = 5$.

the projection of $\boldsymbol{v}$ onto $\mathbb{PP}_d$. Then, our statement is that we can easily find the even integer r such that $\Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ can be expressed as a convex combination of vectors in $\mathbb{PP}_d^r$ and $\mathbb{PP}_d^{r+2}$.

Finally, in Sec. 4.4 we develop our projection algorithm. Roughly, our approach is as follows. Given a vector $\boldsymbol{v} \in \mathbb{R}^d$ we first compute r, the constituent parity of its projection. Given the two-slice representation, projecting onto $\mathbb{PP}_d$ is equivalent to determining an $\alpha \in [0, 1]$, a vector $\boldsymbol{a} \in \mathbb{PP}_d^r$, and a vector $\boldsymbol{b} \in \mathbb{PP}_d^{r+2}$ such that the ℓ_2 norm of $\boldsymbol{v} - \alpha\boldsymbol{a} - (1-\alpha)\boldsymbol{b}$ is minimized.

In [3] we showed that, given α, this projection can be accomplished in two steps. We first project $\boldsymbol{v}$ onto $\alpha\mathbb{PP}_d^r = \{\mathbf{x} \in \mathbb{R}^d | 0 \leq x_i \leq \alpha, \sum_{i=1}^d x_i = \alpha r\}$ a scaled version of $\mathbb{PP}_d^r$, scaled by the convex weighting parameter. Then we project the residual onto $(1-\alpha)\mathbb{PP}_d^r$. The object $\alpha\mathbb{PP}_d^r$ is an ℓ_1 ball with box constraints. Projection onto $\alpha\mathbb{PP}_d^r$ can be done efficiently using a type of waterfilling. Since the function $\min_{\boldsymbol{a} \in \mathbb{PP}_d^r, \boldsymbol{b} \in \mathbb{PP}_d^{r+2}} \|\boldsymbol{v} - \alpha\boldsymbol{a} - (1-\alpha)\boldsymbol{b}\|_2^2$ is convex in α 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.

In contrast to the original approach, in Section 4.4 we develop a far more efficient algorithm that avoids the pair of projections and the search for α. In particular, taking advantage of the convexity in α we use majorization to characterize the convex hull of $\mathbb{PP}_d^r$ and $\mathbb{PP}_d^{r+2}$ in terms of a few linear constraints (inequalities). As projecting onto the parity polytope is equivalent to projecting onto the convex hull of the two slices, we use the characterization to express the projection problem as a quadratic program, and develop an efficient method that directly solves the quadratic program. Avoiding the search over α yields a considerable speed-up over the original approach taken in [3].

10

4.2   Permutation Invariance of the Parity Polytope and Its Consequences

Let us first describe some of the essential features of the parity polytope that are critical to the development of our efficient projection algorithm. First, note the following

Proposition 4.1 $\boldsymbol{u} \in \mathbb{PP}_d$ *if and only if* $\boldsymbol{\Sigma u}$ *is in the parity polytope for every permutation matrix* $\boldsymbol{\Sigma}$.

This proposition follows immediately because the vertex set $\mathbb{P}_d$ is invariant under permutations of the coordinate axes.

Since we will be primarily concerned with projections onto the parity polytope, let us consider the optimization problem

$$\text{minimize}_{\boldsymbol{z}} \|\boldsymbol{v} - \boldsymbol{z}\|_2 \quad \text{subject to } \boldsymbol{z} \in \mathbb{PP}_d . \tag{4.3}$$

The optimal $\boldsymbol{z}^*$ of this problem is the Euclidean projection of $\mathbf{v}$ onto $\mathbb{PP}_d$, which we denote by $\boldsymbol{z}^* = \Pi_{\mathbb{PP}_d}(\boldsymbol{v})$. Again using the symmetric nature of $\mathbb{PP}_d$, we can show the useful fact that if $\boldsymbol{v}$ is sorted in descending order, then so is $\Pi_{\mathbb{PP}_d}(\boldsymbol{v})$.

Proposition 4.2 *Given a vector* $\boldsymbol{v} \in \mathbb{R}^d$, *the component-wise ordering of* $\Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ *is same as that of* $\boldsymbol{v}$.

Proof   We prove the claim by contradiction. Write $\boldsymbol{z}^* = \Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ and suppose that for indices i and j we have $v_i > v_j$ but $z_i^* < z_j^*$. Since all permutations of $\boldsymbol{z}^*$ are in the parity polytope, we can swap components i and j of $\boldsymbol{z}^*$ to obtain another vector in $\mathbb{PP}_d$. Under the assumption $z_j^* > z_i^*$ and $v_i - v_j > 0$ we have $z_j^*(v_i - v_j) > z_i^*(v_i - v_j)$. This inequality implies that $(v_i - z_i^*)^2 + (v_j - z_j^*)^2 > (v_i - z_j^*)^2 + (v_j - z_i^*)^2$, and hence we get that the Euclidean distance between $\boldsymbol{v}$ and $\boldsymbol{z}^*$ is greater than the Euclidean distance between $\boldsymbol{v}$ and the vector obtained by swapping the components.   ∎

These two propositions allow us assume through the remainder of this section that our vectors are presented sorted in descending order unless explicitly stated otherwise.

The permutation invariance of the parity polytope also lets us also employ powerful tools from the theory of majorization to simplify membership testing and projection. The fundamental theorem we exploit is based on the following definition.

Definition 4.3 *Let* $\boldsymbol{u}$ *and* $\boldsymbol{w}$ *be d-vectors sorted in decreasing order. The vector* $\boldsymbol{w}$ *majorizes* $\boldsymbol{u}$ *if*

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

Our results rely on the following Theorem, which states that a vector lies in the convex hull of all permutations of another vector if and only if the former is majorized by the latter (see [28] and references therein).

Theorem 4.4 *Suppose* $\boldsymbol{u}$ *and* $\boldsymbol{w}$ *are d-vectors sorted in decreasing order. Then* $\boldsymbol{u}$ *is in the convex hull of all permutations of* $\boldsymbol{w}$ *if and only if* $\boldsymbol{w}$ *majorizes* $\boldsymbol{u}$.

To gain intuition for why this theorem might hold, suppose that $\boldsymbol{u}$ is in the convex hull of all of the permutations of $\boldsymbol{w}$. Then $\boldsymbol{u} = \sum_{i=1}^{n} p_i \boldsymbol{\Sigma}_i \boldsymbol{w}$ with $\boldsymbol{\Sigma}_i$ being permutation matrices, $p_i \geq 0$, and $\mathbf{1}^T \mathbf{p} = 1$. The matrix $\boldsymbol{Q} = \sum_{i=1}^{n} p_i \boldsymbol{\Sigma}_i$ is doubly stochastic, and one can immediately check that if $\boldsymbol{u} = \boldsymbol{Q}\boldsymbol{w}$ and $\boldsymbol{Q}$ is doubly stochastic, then $\boldsymbol{w}$ majorizes $\boldsymbol{u}$.

To apply majorization theory to the parity polytope, begin with one of the permutahedra $\mathbb{PP}_d^s$. We recall that $\mathbb{PP}_d^s$ is equal to the convex hull of all binary vectors with weight s, equivalently the convex hull of all permutations of the vector consisting of s ones followed by $d - s$ zeros. Thus, by Theorem 4.4, $\boldsymbol{u} \in [0, 1]^d$ is in $\mathbb{PP}_d^s$ if and only if

$$\sum_{k=1}^{q} u_k \leq \min(q, s) \quad \forall \, 1 \leq q < d, \tag{4.4}$$

$$\sum_{k=1}^{d} u_k = s. \tag{4.5}$$

The parity polytope $\mathbb{PP}_d$ is simply the convex hull of all of the $\mathbb{PP}_d^s$ with s even. Thus, we can use majorization to provide an alternative characterization of the parity polytope to that of Yannakakis or Jeroslow.

Lemma 4.5 *A sorted vector $\boldsymbol{u} \in \mathbb{PP}_d$ if and only if there exist non-negative coefficients $\{\mu_s\}_{even \, s \leq d}$ such that*

$$\sum_{s \text{ even}}^{d} \mu_s = 1, \quad \mu_s \geq 0. \tag{4.6}$$

$$\sum_{k=1}^{q} u_k \leq \sum_{s \text{ even}}^{d} \mu_s \min(q, s) \quad \forall \, 1 \leq q < d \tag{4.7}$$

$$\sum_{k=1}^{d} u_k = \sum_{s \text{ even}}^{d} \mu_s s. \tag{4.8}$$

Proof First, note that every vertex of $\mathbb{PP}_d$ of weight s satisfies these inequalities with $\mu_s = 1$ and $\mu_{s'} = 0$ for $s' \neq s$. Thus $\boldsymbol{u} \in \mathbb{PP}_d$ must satisfy (4.6)-(4.8). Conversely, if $\boldsymbol{u}$ satisfies (4.6)-(4.8), then $\boldsymbol{u}$ is majorized by the vector

$$\boldsymbol{w} = \sum_{s \text{ even}}^{d} \mu_s \boldsymbol{b}_s$$

where $\boldsymbol{b}_s$ is a vector consisting of s ones followed by $d - s$ zeros. $\boldsymbol{w}$ is contained in $\mathbb{PP}_d$ as are all of its permutations. Thus, we conclude that $\boldsymbol{u}$ is also contained in $\mathbb{PP}_d$. ∎

While Lemma 4.5 characterizes the containment of a vector in $\mathbb{PP}_d$, the relationship is not one-to-one; for a particular $\boldsymbol{u} \in \mathbb{PP}_d$ there can be many sets $\{\mu_s\}$ that satisfy the lemma. We will next show that there is always one assignment of μ_s with only two non-zero μ_s.

4.3 Constituent Parity of the Projection

For $a \in \mathbb{R}$, let $\lfloor a \rfloor_{\text{even}}$ denote the "even-floor" of a, i.e., the largest even integer r such that $r \le a$. Define the "even-ceiling," $\lceil a \rceil_{\text{even}}$ similarly. For a vector $\boldsymbol{u}$ we term $\lfloor \|\boldsymbol{u}\|_1 \rfloor_{\text{even}}$ the *constituent parity* of vector $\boldsymbol{u}$. In this section we will show that if $\boldsymbol{u} \in \mathbb{PP}_d$ has constituent parity r, then it can be written as a convex combination of binary vectors with weight equal to r and $r + 2$. This result is summarized by the following

Lemma 4.6 *("Two-slice" lemma) A vector $\boldsymbol{u} \in \mathbb{PP}_d$ iff $\boldsymbol{u}$ can be expressed as a convex combination of vectors in $\mathbb{PP}_d^r$ and $\mathbb{PP}_d^{r+2}$ where $r = \lfloor \|\boldsymbol{u}\|_1 \rfloor_{\text{even}}$.*

Proof Consider any (sorted) $\boldsymbol{u} \in \mathbb{PP}_d$. Lemma 4.5 tells us that there is always (at least one) set $\{\mu_s\}$ that satisfy (4.6)–(4.8). Letting r be defined as in the lemma statement, we define α to be the unique scalar between zero and one that satisfies the relation $\|\boldsymbol{u}\|_1 = \alpha r + (1 - \alpha)(r + 2)$:

$$\alpha = \frac{2 + r - \|\boldsymbol{u}\|_1}{2}. \tag{4.9}$$

Then, we choose the following candidate assignment: $\mu_r = \alpha$, $\mu_{r+2} = 1 - \alpha$, and all other $\mu_s = 0$. We show that this choice satisfies (4.6)–(4.8) which will in turn imply that there is a $\boldsymbol{u}_r \in \mathbb{PP}_d^r$ and a $\boldsymbol{u}_{r+2} \in \mathbb{PP}_d^{r+2}$ such that $\boldsymbol{u} = \alpha \boldsymbol{u}_r + (1 - \alpha) \boldsymbol{u}_{r+2}$.

First, by the definition of α, (4.6) and (4.8) are both satisfied. Further, for the candidate set the relations (4.7) and (4.8) simplify to

$$\sum_{k=1}^{q} u_k \le \alpha \min(q, r) + (1 - \alpha) \min(q, r + 2), \ \forall \ 1 \le q < d, \tag{4.10}$$

$$\sum_{k=1}^{d} u_k = \alpha r + (1 - \alpha)(r + 2). \tag{4.11}$$

To show that (4.10) is satisfied is straightforward for the cases $q \le r$ and $q \ge r+2$. First consider any $q \le r$. Since $\min(q, r) = \min(q, r + 2) = q$, $u_k \le 1$ for all k, and there are only q terms, (4.10) must hold. Second, consider any $q \ge r+2$. We use (4.11) to write $\sum_{k=1}^{q} u_k = \alpha r + (1-\alpha)(r+2) - \sum_{q+1}^{d} u_k$. Since $u_k \ge 0$ this is upper bounded by $\alpha r + (1 - \alpha)(r + 2)$ which we recognize as the right-hand side of (4.10) since $r = \min(q, r)$ and $r + 2 = \min(q, r + 2)$.

It remains to verify only one more inequality in (4.10) namely the case when $q = r + 1$, which is

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

To show that the above inequality holds, we maximize the right-hand-side of (4.7) across *all* valid choices of $\{\mu_s\}$ and show that the resulting maximum is exactly $r + 1 - \alpha$. Since this maximum is attainable by some choice of $\{\mu_s\}$ and our choice meets that bound, our choice is a valid choice.

The logic is as follows. Since $\boldsymbol{u} \in \mathbb{PP}_d$ any valid choice for $\{\mu_s\}$ must satisfy (4.6) which, for $q = r + 1$, is

$$\sum_{k=1}^{r+1} u_k \le \sum_{s \text{ even}}^{d} \mu_s \min(s, r + 1). \tag{4.12}$$

13

To see that across *all* valid choice of $\{\mu_s\}$ the largest value attainable for the right hand side is precisely $r + 1 - \alpha$ consider the linear program

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

The first two constraints are simply (4.6) and (4.8). Recognizing $\alpha r + (1-\alpha)(r+2) = r + 2 - 2\alpha$, the dual program is

$$
\begin{array}{ll}
\text{minimize} & (r + 2 - 2\alpha)\lambda_1 + \lambda_2 \\
\text{subject to} & \lambda_1 s + \lambda_2 \geq \min(s, r+1) \ \forall \ s \text{ even}.
\end{array}
$$

Setting $\mu_r = \alpha$, $\mu_{r+2} = (1 - \alpha)$, the other primal variable to zero, $\lambda_1 = 1/2$, and $\lambda_2 = r/2$, 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 (4.12) is at most $r + 1 - \alpha$.

We have proved that if $\boldsymbol{u} \in \mathbb{PP}_d$ then the choice of $r = \lfloor \|\boldsymbol{u}\|_1 \rfloor_{\text{even}}$ and α as in (4.9) satisfies the requirements of Lemma 4.5 and so we can express $\boldsymbol{u}$ as $\boldsymbol{u} = \alpha \boldsymbol{u}_r + (1 - \alpha)\boldsymbol{u}_{r+2}$. The converse—given a vector $\boldsymbol{u}$ that is a convex combination of vectors in $\mathbb{PP}_d^r$ and $\mathbb{PP}_d^{r+2}$ it is in $\mathbb{PP}_d$—holds because conv$(\mathbb{PP}_d^r \cup \mathbb{PP}_d^{r+2}) \subseteq \mathbb{PP}_d$. ∎

A useful consequence of Theorem 4.4 is the following corollary.

Corollary 4.7 *Let $\boldsymbol{u}$ be a vector in $[0, 1]^d$. If $\sum_{k=1}^d u_k$ is an even integer then $\boldsymbol{u} \in \mathbb{PP}_d$.*

Proof Let $\sum_{k=1}^d u_k = s$. Since $\boldsymbol{u}$ is majorized by a sorted binary vector of weight s then, by Theorem 4.4, $\boldsymbol{u} \in \mathbb{PP}_d^s$ which, in turn, implies $\boldsymbol{u} \in \mathbb{PP}_d$. ∎

We conclude this section by showing that we can easily compute the constituent parity of $\Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ without explicitly computing the projection of $\boldsymbol{v}$.

Lemma 4.8 *For any vector $\boldsymbol{v} \in \mathbb{R}^d$, let $\boldsymbol{z} = \Pi_{[0,1]^d}(\boldsymbol{v})$, the projection of $\boldsymbol{v}$ onto $[0, 1]^d$ and denote by $\Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ the projection of $\boldsymbol{v}$ onto the parity polytope. Then*

$$
\lfloor \|\boldsymbol{z}\|_1 \rfloor_{even} \leq \|\Pi_{\mathbb{PP}_d}(\boldsymbol{v})\|_1 \leq \lceil \|\boldsymbol{z}\|_1 \rceil_{even} .
$$

That is, we can compute the constituent parity of the projection of $\boldsymbol{v}$ by projecting $\boldsymbol{v}$ onto $[0, 1]^d$ and computing the even floor.

Proof Let $\rho_U = \lceil \|\boldsymbol{z}\|_1 \rceil_{\text{even}}$ and $\rho_L = \lfloor \|\boldsymbol{z}\|_1 \rfloor_{\text{even}}$. We prove the following fact: given any $\boldsymbol{y}' \in \mathbb{PP}_d$ with $\|\boldsymbol{y}'\|_1 > \rho_U$ there exits a vector $\boldsymbol{y} \in [0, 1]^d$ such that $\|\boldsymbol{y}\|_1 = \rho_U$, $\boldsymbol{y} \in \mathbb{PP}_d$, and $\|\boldsymbol{v} - \boldsymbol{y}\|_2^2 < \|\boldsymbol{v} - \boldsymbol{y}'\|_2^2$. 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 $\boldsymbol{v}$. 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 $\boldsymbol{y}$ based on $\boldsymbol{y}'$ and $\boldsymbol{z}$. Define the set of "high" values to be the coordinates on which y_i' is greater than z_i, i.e., $\mathcal{H} := \{i \in [d] \mid y_i' > z_i\}$. Since by assumption $\|\boldsymbol{y}'\|_1 > \rho_U \geq \|\boldsymbol{z}\|_1$ we know that $|\mathcal{H}| \geq 1$. Consider the test vector $\boldsymbol{t}$ defined component-wise as

$$
t_i = \begin{cases} z_i & \text{if } i \in \mathcal{H}, \\ y_i' & \text{otherwise}. \end{cases}
$$

Note that $\|\boldsymbol{t}\|_1 \leq \|\boldsymbol{z}\|_1 \leq \rho_U < \|\boldsymbol{y}'\|_1$. The vector $\boldsymbol{t}$ differs from $\boldsymbol{y}'$ only in $\mathcal{H}$. Thus, by changing (reducing) components of $\boldsymbol{y}'$ in the set $\mathcal{H}$ we can obtain a vector $\boldsymbol{y}$ such that $\|\boldsymbol{y}\|_1 = \rho_U$. In particular there exists a vector $\boldsymbol{y}$ with $\|\boldsymbol{y}\|_1 = \rho_U$ such that $y_i' \geq y_i \geq z_i$ for $i \in \mathcal{H}$ and $y_i = y_i'$ for $i \notin \mathcal{H}$. Since the ℓ_1 norm of $\boldsymbol{y}$ is even and it is in $[0,1]^d$ we have by Corollary 4.7 that $\boldsymbol{y} \in \mathbb{PP}_d$.

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

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

4.4 Projection Algorithm

In this section we formulate a quadratic program (Problem PQP) for the projection problem and then develop an algorithm (Algorithm 2) that efficiently solves the quadratic program.

Given a vector $\boldsymbol{v} \in \mathbb{R}^d$, set $r = \lfloor \|\Pi_{[0,1]^d}(\boldsymbol{v})\|_1 \rfloor_{\text{even}}$. From Lemma 4.8 we know that the constituent parity of $\boldsymbol{z}^* := \Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ is r. We also know that $\boldsymbol{z}^*$ is sorted in descending order if $\boldsymbol{v}$ is. Let $\boldsymbol{S}$ be a $(d-1) \times d$ matrix with diagonal entries set to 1, $\boldsymbol{S}_{i,i+1} = -1$ for $1 \leq i \leq d-1$, and zero everywhere else:

$$\boldsymbol{S} = \begin{bmatrix} 1 & -1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & -1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & -1 & \ldots & 0 & 0 \\ \vdots & & & \ddots & \ddots & & \vdots \\ 0 & 0 & 0 & 0 & \ldots & -1 & 0 \\ 0 & 0 & 0 & 0 & \ldots & 1 & -1 \end{bmatrix}.$$

The constraint that $\boldsymbol{z}^*$ has to be sorted in decreasing order can be stated as $\boldsymbol{S}\boldsymbol{z}^* \geq \boldsymbol{0}$, where $\boldsymbol{0}$ is the all-zeros vector.

In addition, Lemma 4.6 implies that $\boldsymbol{z}^*$ is a convex combination of vectors of Hamming weight r and $r+2$. Using inequality (4.10) we get that a d-vector $\boldsymbol{z} \in [0,1]^d$, with

$$\sum_{i=1}^d z_i = \alpha r + (1-\alpha)(r+2), \tag{4.13}$$

is a convex combination of vectors of weight r and $r+2$ iff it satisfies the following bounds:

$$\sum_{k=1}^q z_{(k)} \leq \alpha \min(q,r) + (1-\alpha)\min(q,r+2) \quad \forall \ 1 \leq q < d, \tag{4.14}$$

where $z_{(k)}$ denotes the kth largest component of $\boldsymbol{z}$. As we saw in the proof of Lemma 4.5, the fact that the components of $\boldsymbol{z}$ are no more than one implies that inequalities (4.14) are satisfied

for all $q \leq r$. Also, (4.13) enforces the inequalities for $q \geq r + 2$. Therefore, inequalities in (4.14) for $q \leq r$ and $q \geq r + 2$ are redundant. Note that in addition we can eliminate the variable α by solving (4.13) giving $\alpha = 1 + \frac{r - \sum_{k=1}^{d} z_k}{2}$ (see also (4.9)). Therefore, for a sorted vector $\boldsymbol{v}$, we can write the projection onto $\mathbb{PP}_d$ as the optimization problem

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{v} - \boldsymbol{z}\|_2^2$$
$$\text{subject to} \quad 0 \leq z_i \leq 1 \ \ \forall \ i$$
$$\boldsymbol{S}\boldsymbol{z} \geq 0$$

$$0 \leq 1 + \frac{r - \sum_{k=1}^{d} z_k}{2} \leq 1 \tag{4.15}$$

$$\sum_{k=1}^{r+1} z_k \leq r - \frac{r - \sum_{k=1}^{d} z_k}{2}. \tag{4.16}$$

The last two constraints can be simplified as follows. First, constraint (4.15) simplifies to $r \leq \sum_{k=1}^{d} z_k \leq r + 2$. Next, defining the vector

$$\boldsymbol{f}_r = (\underbrace{1, 1, \ldots, 1}_{r+1}, \underbrace{-1, -1, \ldots, -1}_{d-r-1})^T. \tag{4.17}$$

we can rewrite inequality (4.16) as $\boldsymbol{f}_r^T \boldsymbol{z} \leq r$. Using these simplifications yields the final form of our quadratic program:

Problem PQP:

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{v} - \boldsymbol{z}\|_2^2$$
$$\text{subject to} \quad 0 \leq z_i \leq 1 \ \ \forall \ i \tag{4.18}$$
$$\boldsymbol{S}\boldsymbol{z} \geq \boldsymbol{0} \tag{4.19}$$
$$r \leq \boldsymbol{1}^T \boldsymbol{z} \leq r + 2 \tag{4.20}$$
$$\boldsymbol{f}_r^T \boldsymbol{z} \leq r. \tag{4.21}$$

The projection algorithm we develop efficiently solves the KKT conditions of PQP. The objective function is strongly convex and the constraints are linear. Hence, the KKT conditions are not only necessary but also sufficient for optimality. To formulate the KKT conditions, we first construct the Lagrangian with dual variables β, $\boldsymbol{\mu}$, $\boldsymbol{\gamma}$, ξ, $\boldsymbol{\theta}$, and ζ:

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{v} - \boldsymbol{z}\|_2^2 - \beta \left(r - \boldsymbol{f}_r^T \boldsymbol{z}\right) - \boldsymbol{\mu}^T(\boldsymbol{1} - \boldsymbol{z}) - \boldsymbol{\gamma}^T \boldsymbol{z}$$
$$- \xi \left(r + 2 - \boldsymbol{1}^T \boldsymbol{z}\right) - \zeta(\boldsymbol{1}^T \boldsymbol{z} - r) - \boldsymbol{\theta}^T \boldsymbol{S}\boldsymbol{z}.$$

The KKT conditions are then given by stationarity of the Lagrangian, complementary slackness,

and feasibility.

$$\boldsymbol{z} = \boldsymbol{v} - \beta \boldsymbol{f}_r - \boldsymbol{\mu} + \boldsymbol{\gamma} - (\xi - \zeta)\mathbf{1} + \boldsymbol{S}^T \boldsymbol{\theta}. \tag{4.22}$$

$$0 \le \beta \quad \perp \quad \boldsymbol{f}_r^T \boldsymbol{z} - r \le 0$$
$$\mathbf{0} \le \boldsymbol{\mu} \quad \perp \quad \boldsymbol{z} \le \mathbf{1}$$
$$\mathbf{0} \le \boldsymbol{\gamma} \quad \perp \quad \boldsymbol{z} \ge \mathbf{0}$$
$$\mathbf{0} \le \boldsymbol{\theta} \quad \perp \quad \boldsymbol{S}\boldsymbol{z} \ge \mathbf{0}$$
$$0 \le \xi \quad \perp \quad \mathbf{1}^T \boldsymbol{z} - r - 2 \le 0$$
$$0 \le \zeta \quad \perp \quad \mathbf{1}^T \boldsymbol{z} - r \ge 0.$$

A vector $\boldsymbol{z}$ that satisfies (4.22) and the following orthogonality conditions is equal to the projection of $\boldsymbol{v}$ onto $\mathbb{PP}_d$.

To proceed, set $\beta_{\max} = \frac{1}{2}[v_{r+1} - v_{r+2}]$ and define the parameterized vector

$$\boldsymbol{z}(\beta) := \Pi_{[0,1]^d}(\boldsymbol{v} - \beta \boldsymbol{f}_r). \tag{4.23}$$

The following lemma implies that the optimizer of PQP, i.e., $\boldsymbol{z}^* = \Pi_{\mathbb{PP}_d}(\boldsymbol{v})$, is $\boldsymbol{z}(\beta_{\mathrm{opt}})$ for some $\beta_{\mathrm{opt}} \in [0, \beta_{\max}]$.

Lemma 4.9 *There exists a $\beta_{opt} \in [0, \beta_{max}]$ such that $\boldsymbol{z}(\beta_{opt})$ satisfies the KKT conditions of the quadratic program PQP.*

Proof Note that when $\beta > \beta_{\max}$ we have that $z_{r+1}(\beta) < z_{r+2}(\beta)$ and $\boldsymbol{z}(\beta)$ is ordered differently from $\boldsymbol{v}$ and $\boldsymbol{f}_r^T \boldsymbol{z}(\beta) < r$. Consequently $\boldsymbol{z}(\beta)$ cannot be the projection onto $\mathbb{PP}_d$ for $\beta > \beta_{\max}$. At the other boundary of the interval, when $\beta = 0$ we have $\boldsymbol{z}(0) = \Pi_{[0,1]^d}(\boldsymbol{v})$. If $\boldsymbol{f}_r^T \boldsymbol{z}(0) = r$, then $\boldsymbol{z}(0) \in \mathbb{PP}_d$ by Corollary 4.7. But since $\boldsymbol{z}(0)$ is the closest point in $[0,1]^d$ to $\boldsymbol{v}$, it must also be the closest point in $\mathbb{PP}_d$.

Assume now that $\boldsymbol{f}_r^T \boldsymbol{z}(0) > r$. Taking the directional derivative with respect to β increasing, we obtain the following:

$$\frac{\partial \boldsymbol{f}_r^T \boldsymbol{z}(\beta)}{\partial \beta} = \boldsymbol{f}_r^T \frac{\partial \boldsymbol{z}(\beta)}{\partial \beta}$$
$$= \sum_{k: \ 0 < z_k(\beta) < 1} -f_{r,k}^2$$
$$= -\big|\{k \mid 1 \le k \le d, 0 < z_k(\beta) < 1\}\big| \tag{4.24}$$
$$< 0.$$

proving that $\boldsymbol{f}_r^T \boldsymbol{z}(\beta)$ is a decreasing function of β. Therefore, by the mean value theorem, there exists a $\beta_{\mathrm{opt}} \in [0, \beta_{\max}]$ such that $\boldsymbol{f}_r^T \boldsymbol{z}(\beta_{\mathrm{opt}}) = r$.

First note that $\boldsymbol{z}(\beta_{\mathrm{opt}})$ is feasible for Problem PQP. We need only verify (4.20). Recalling that r is defined as $r = \lfloor \|\Pi_{[0,1]^d}(\boldsymbol{v})\|_1 \rfloor_{\mathrm{even}}$, we get the lower bound:

$$\mathbf{1}^T \boldsymbol{z}(\beta_{\mathrm{opt}}) \ge \boldsymbol{f}_r^T \boldsymbol{z}(\beta_{\mathrm{opt}}) = r.$$

The components of $\boldsymbol{z}(\beta_{\mathrm{opt}})$ are all less than one, so $\sum_{k=1}^{r+1} z_k(\beta_{\mathrm{opt}}) \le r + 1$. Combining this with the equality $\boldsymbol{f}_r^T \boldsymbol{z}(\beta_{\mathrm{opt}}) = r$ tells us that $\sum_{k=r+2}^{d} z_k(\beta_{\mathrm{opt}}) \le 1$. We therefore find that $\mathbf{1}^T \boldsymbol{z}(\beta_{\mathrm{opt}})$ is no more than $r + 2$.

To complete the proof, we need only find dual variables to certify the optimality. Setting ξ, ζ, and θ to zero, and μ and γ to the values required to satisfy (4.22) provides the necessary assignments to satisfy the KKT conditions. ∎

Lemma 4.9 thus certifies that all we need to do to compute the projection is to compute the optimal β. To do so, we use the fact that the function $\boldsymbol{f}_r^T \boldsymbol{z}(\beta)$ is a piecewise linear function of β. For a fixed β, define the *active set* to be the indices where $\boldsymbol{z}(\beta)$ is strictly between 0 and 1

$$\mathcal{A}(\beta) := \{k \mid 1 \le k \le d, 0 < z_k(\beta) < 1\}. \tag{4.25}$$

Let the *clipped set* be the indices where $\boldsymbol{z}(\beta)$ is equal to 1.

$$\mathcal{C}(\beta) := \{k \mid 1 \le k \le d, z_k(\beta) = 1\}. \tag{4.26}$$

Let the *zero set* be the indices where $\boldsymbol{z}(\beta)$ is equal to zero

$$\mathcal{Z}(\beta) := \{k \mid 1 \le k \le d, z_k(\beta) = 0\}. \tag{4.27}$$

Note that with these definitions, we have

$$\boldsymbol{f}_r^T \boldsymbol{z}(\beta) = |\mathcal{C}(\beta)| + \sum_{j \in \mathcal{A}(\beta)} (z_j - \beta)$$
$$= |\mathcal{C}(\beta)| - \beta|\mathcal{A}(\beta)| + \sum_{j \in \mathcal{A}(\beta)} z_j \tag{4.28}$$

Our algorithm simply increases beta until the active set changes, keeping track of the sets $\mathcal{A}(\beta)$, $\mathcal{C}(\beta)$, and $\mathcal{Z}(\beta)$. We break the interval $[0, \beta_{\max}]$ into the locations where the active set changes, and compute the value of $\boldsymbol{f}_r^T \boldsymbol{z}(\beta)$ at each of these breakpoints until $\boldsymbol{f}_r^T \boldsymbol{z}(\beta) < r$. At this point, we have located the appropriate active set for optimality and can find β_{opt} by solving the linear equation (4.28).

The breakpoints themselves are easy to find: they are the values of β where an index is set equal to one or equal to zero. First, define the following sets

$$\begin{aligned}
\mathcal{B}_1 &:= \{v_i - 1 \mid 1 \le i \le r + 1\}, \\
\mathcal{B}_2 &:= \{v_i \mid 1 \le i \le r + 1\}, \\
\mathcal{B}_3 &:= \{-v_i \mid r + 2 \le i \le d\}, \\
\mathcal{B}_4 &:= \{-v_i + 1 \mid r + 2 \le i \le d\}.
\end{aligned}$$

The sets $\mathcal{B}_1$ and $\mathcal{B}_2$ concern the $r + 1$ largest components of $\boldsymbol{v}$; $\mathcal{B}_3$ and $\mathcal{B}_4$ the smallest components. The set of breakpoints is

$$\mathcal{B} := \left\{ \beta \in \bigcup_{j=1}^{4} \mathcal{B}_j \;\middle|\; 0 \le \beta \le \beta_{\max} \right\} \cup \{0, \beta_{\max}\}.$$

There are thus at most $2d + 2$ breakpoints.

To summarize, our Algorithm 2 sorts the input vector, computes the set of breakpoints, and then marches through the breakpoints until it finds a value of $\beta_i \in \mathcal{B}$ with $\boldsymbol{f}_r^T \boldsymbol{z}(\beta_i) \le r$. Since we
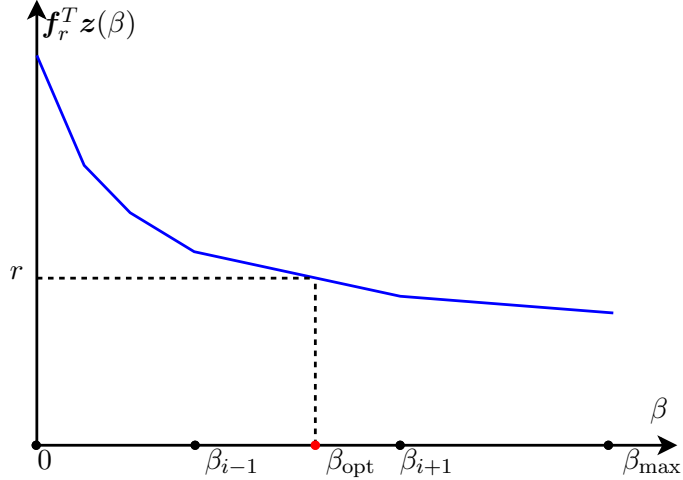
Figure 2: Since there are a finite number of breakpoints (at most $2d + 2$) and the function $\boldsymbol{f}_r^T \boldsymbol{z}(\beta)$ is linear between breakpoints, we can solve for β_{opt} in linear time. See (4.17) and (4.23) for definitions of $\boldsymbol{f}_r$ and $\boldsymbol{z}(\beta)$ respectively.

will also have $\boldsymbol{f}_r^T \boldsymbol{z}(\beta_{i-1}) > r$, the optimal β will lie in $[\beta_{i-1}, \beta_i]$ and can be found by solving (4.28). In the algorithm box for Algorithm 2, b is the largest and a is the smallest index in the active set. We use V to denote the sum of the elements in the active set and Λ the total sum of the vector at the current break point. Some of the awkward **if** statements in the main **for** loop take care of the cases when the input vector has many repeated entries.

Algorithm 2 requires two sorts (sorting the input vector and sorting the breakpoints), and then an inspection of at most $2d$ breakpoints. Thus, the total complexity of the algorithm is linear plus the time for the two sorts.

5 Numerical results and implementation

In this section, we present simulation results for the ADMM decoder and discuss various aspects of our implementation. In Section 5.1 we present word-error-rate (WER) results for two LDPC codes. In Section 5.2 we discuss how the various parameters choices in ADMM affect decoding performance, as measured by error rate and by decoding time.

5.1 Error-rate performance

In this section we present WER results for the ADMM decoder. We present simulation results for two codes and compare to sum-product BP decoding. We simulate both codes over the AWGN channel with binary inputs. The first code is the [2640, 1320] rate-0.5, (3, 6)-regular Margulis LDPC code [35]. The second is a [1057, 244] rate-0.77, (3, 13)-regular LDPC code obtained from [26]. This code is also studied by Yedidia *et al.* [55]. We choose both codes as they have been chosen in the past to study error-floor performance.

In Fig. 3 we plot the WER performance of the Margulis code for the ADMM decoder and various implementations of sum-product BP decoding. As mentioned, this code has been extensively studied in the literature due to its error floor behavior (see, e.g., [8, 27, 35]). Recently it has been

Algorithm 2 Given $\boldsymbol{u} \in \mathbb{R}^d$ determine its projection on $\mathbb{PP}_d$, $\boldsymbol{z}^*$

1: Permute $\boldsymbol{u}$ to produce a vector $\boldsymbol{v}$ whose components are sorted in decreasing order, i.e., $v_1 \geq v_2 \geq \ldots \geq v_d$. Let $\boldsymbol{Q}$ be the corresponding permutation matrix, i.e., $\boldsymbol{v} = \boldsymbol{Q}\boldsymbol{u}$.

2: Compute $\hat{\boldsymbol{z}} \leftarrow \Pi_{[0,1]^d}(\boldsymbol{v})$.

3: Assign $r = \lfloor \|\hat{\boldsymbol{z}}\|_1 \rfloor_{\text{even}}$ and $\beta_{\max} = \frac{1}{2}[\hat{z}_{r+1} - \hat{z}_{r+2}]$.

4: Define $\boldsymbol{f}_r$ as in (4.17).

5: **if** $\boldsymbol{f}_{r+1}^T \hat{\boldsymbol{z}} \leq r$ **then**

6:    Return $\boldsymbol{z}^* = \hat{\boldsymbol{z}}$.

7: **end if**

8: Assign $\mathcal{E}_1 = \{v_i - 1 \mid 1 \leq i \leq r+1\}$,
$\mathcal{L}_1 = \{v_i \mid 1 \leq i \leq r+1\}$,
$\mathcal{E}_2 = \{-v_i \mid r+2 \leq i \leq d\}$,
$\mathcal{L}_2 = \{-v_i + 1 \mid r+2 \leq i \leq d\}$.

9: Assign the set of breakpoints:
$\mathcal{B} := \left\{ \beta \in \cup_{j=1}^2 \mathcal{E}_j \cup \mathcal{L}_j) \mid 0 \leq \beta \leq \beta_{\max} \right\} \cup \{0, \beta_{\max}\}$.

10: Index the breakpoints in $\mathcal{B}$ in a sorted manner to get $\{\beta_i\}_i$ where $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_{|\mathcal{B}|}$.

11: Initialize a as the smallest index such that $0 < \hat{z}_a < 1$.

12: Initialize b as the largest index such that $0 < \hat{z}_b < 1$.

13: Initialize sum $V = \boldsymbol{f}_r^T \hat{\boldsymbol{z}}$.

14: **for** $i = 1$ to $|\mathcal{B}|$ **do**

15:    Set $\beta_0 \leftarrow \beta_i$.

16:    **if** $\beta_i \in \mathcal{E}_1 \cup \mathcal{E}_2$ **then**

17:       Update $a \leftarrow a - 1$.

18:       Update $V \leftarrow V + v_a$.

19:    **else**

20:       Update $b \leftarrow b + 1$

21:       Update $V \leftarrow V - v_b$.

22:    **end if**

23:    **if** $i < d$ and $\beta_i \neq \beta_{i+1}$ **then**

24:       $\Lambda \leftarrow (a - 1) + V - \beta_0(b - a + 1)$

25:       **if** $\Lambda \leq r$ **then break**

26:    **else if** $i = d$ **then**

27:       $\Lambda \leftarrow (a - 1) + V - \beta_0(b - a + 1)$

28:    **end if**

29: **end for**

30: **if** $\Lambda > r$ **then**

31:    Compute $\beta_{\text{opt}} \leftarrow \beta - \frac{r - \Lambda}{b - a + 1}$.

32: **else**

33:    $\beta_0 \leftarrow \beta_{i-1}$

34:    $a \leftarrow |\{j \mid v_j - \beta > 1\}|$

35:    $b \leftarrow r + 2 + |\{j \mid v_j + \beta \geq 0\}|$

36:    $V \leftarrow \sum_{j=b+1}^{r+1} v_j - \sum_{j=r+2}^{a-1} v_j$

37:    $\beta_{\text{opt}} \leftarrow \frac{r - b - V}{b - a + 1)}$

38: **end if**

39: Return $\boldsymbol{z}^* = \boldsymbol{Q}^T \Pi_{[0,1]^d}(\boldsymbol{z} - \beta_{\text{opt}} \boldsymbol{f}_r)$.
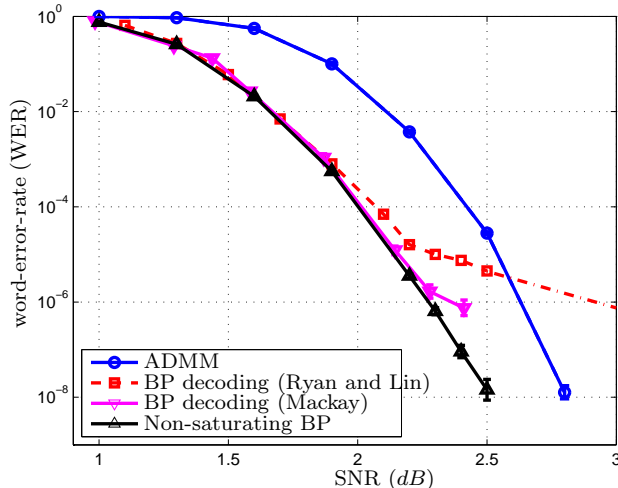
Figure 3: Word error rate (WER) of the $[2540, 1320]$ "Margulis" LDPC code used on the AWGN channel plotted as a function of signal-to-noise ratio (SNR). The WER performance of ADMM is compared to that of non-saturating sum-product BP, as well as to results for (saturating) sum-product BP from Ryan and Lin [35] and from MacKay and Postol [27].

noted [8, 9] that the previously observed error floor of this code is, at least partially, a result of saturation in the message LLRs passed by the BP decoder. This issue of implementation can be greatly mitigated by improving the way large LLRs are handled. Thus, alongside these previous results we plot results of our own "non-saturating" sum-product BP implementation, which follows the implementation of [8,9], and which matches the results reported therein. In our simulations of the ADMM decoder we collect more than 200 errors for all data points other than the highest SNR (SNR $= 2.8dB$), for which we collected 10 errors.

The first aspect to note is that the LP decoder has a waterfall behavior, but it occurs at a slightly higher SNR (about $0.4dB$ in this example) than that of sum-product BP. This observation is consistent with earlier simulations of LP decoding for long block lengths, e.g., those presented in [49, 55]. It is worth mentioning that it was show in [52] that fixed points of sum-product BP correspond to stationary points of the Bethe approximation of the free energy when the temperature parameter $T = 1$. However, when the temperature parameter in the Bethe approximation is reduced to $T = 0$ minimizing the Bethe free energy is the same as LP decoding, see, e.g., [41]. While the objective function in sum-product and LP is thus quite different, both optimization problems are subject to the same set of constraints – the "local marginal polytope" (equivalent to the fundamental polytope for LP decoding of binary codes, see [41] for details). Since the objective functions are different, one should not expect identical performance, as the simulations demonstrate.

The second aspect to note is that, as in the prior work, we do not observe an error floor in LP decoding. Considering decoding of this code using the non-saturating version of sum-product we do not observe an abrupt error floor. However, we do see that at WERs of 10^{-8} the waterfall of ADMM is continuing to steepen, while that of sum-product BP appears to be dropping at a constant slope. In this regime we found that the non-saturating BP decoder is not converging to a trapping set, but is rather oscillating, as discussed in [57] [34]. However, as we see in our next
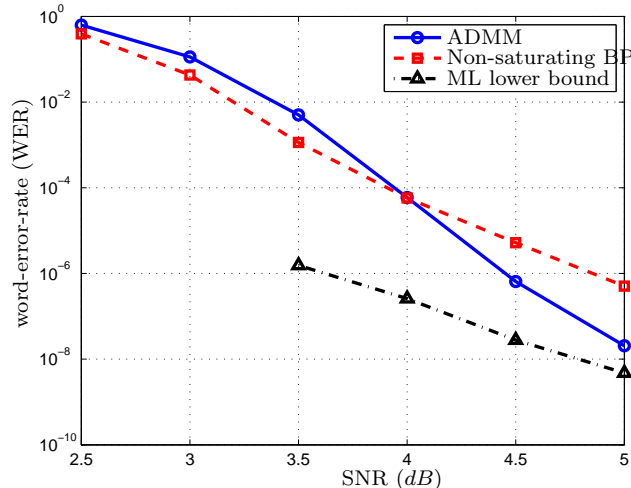
Figure 4: Word error rate (WER) of the $[1057, 244]$ LDPC code used on the AWGN channel plotted as a function of signal-to-noise ratio (SNR). The WER performance of ADMM is compared to that of non-saturating sum-product BP, as well as to an estimated lower-bound on ML decoding.

example, there are codes for which ADMM does not display an error floor while non-saturating sum-product BP does.

Figure 4 presents simulation results for the rate-0.77 length-1057 code. In this simulation, all data points are based on more than 200 errors except for the ADMM data at SNR = 5dB, where 29 errors are observed. In addition we plot an estimated lower bound on maximum likelihood (ML) decoding performances. The lower bound is estimated in the following way. In the ADMM decoding simulations we round any non-integer solution obtained from the ADMM decoder to produce a codeword estimate. If the decoder produces a decoding error, i.e., if the estimate does not match the transmitted codeword, we check if the estimate is a valid codeword. If the estimate satisfies all the parity checks (and is therefore a codeword) we also compare the probability of the estimate given the channel observations with the that of the transmitted codeword given the channel observations. If the probability of estimate is greater than that of the transmitted codeword we know that an ML decoder would also be in error. All other events are counted as ML successes (hence the estimated *lower* bound on ML performance). In contrast to the Margulis code, Fig. 4 shows that for this code the ADMM decoder displays no signs of an error floor, while the BP decoder does. Further, ADMM is approaching the ML error lower bound at high SNRs.

Given the importance of error-floor effects in high reliability applications, and the contrasting outcomes of our simulations, we now make some observations. One point demonstrated by these experiments, in particular by the simulation of the Margulis code, (and argued in [8, 9]) is that numerical precision effects can dramatically affect code performance in the high SNR regime. When precision is limited, a set of incorrect (flipped) channel symbols connected together in a "trapping set" can enforce each others' incorrect beliefs sufficiently so as to outweigh the correct evidence from the rest of the code. Even though the rest of the code symbols may be much more certain of themselves, particularly if the magnitude of beliefs are limited, those limits can prevent the correct

variables from having sufficient influence on the symbols in the trapping set to correct them. From a practical point of view, a real-world implementation would use fixed precision arithmetic. Thus, understanding decoding behavior under finite precision is extremely important.

A second point made by comparing the two codes is that the performance of an algorithm, e.g., non-saturating BP, can vary dramatically from code to code (Margulis vs. 1057) and the performance of a code from algorithm to algorithm (BP vs. ADMM). For each algorithm we might think about three types of codes [54]. The first (type-A) would consist of codes that do not have any trapping sets, i.e., do not display an error floor, even for low-precision implementations. The second (type-B) would consist of codes whose behavior changes with precision (e.g., the Margulis code). The final (type-C) would consist of codes that have trapping sets even under infinite precision (the length-1057 code may belong to this set). Under this taxonomy there are two natural strategies to pursue. The first is to design codes that fall in the first class. This is the approach taken in, e.g., [32] [17] [20] [30] [47], where codes of large-girth are sought. The second is to design improved algorithms that enlarge the set of codes that fall into the first class. This is the approach taken in this paper. Since the ADMM decoder has rigorous convergence guarantees, since ADMM has historically be observed to be quite robust to parameter choices and precision settings, and since the "messages" passed in ADMM (the replica values) are inherently bounded to the unit interval (since the parity polytope is contained within the unit hypercube), we expect that the ADMM decoder will be a strong competitor to BP in applications that demand ultra-high reliabilities.

5.2 Parameter choices

In the ADMM decoding algorithm there are a number of parameters that need to be set. The first is the stopping tolerance, ϵ, the second is the penalty parameter, μ, and the third is the maximum allowable number of iterations, $t_{\max}$. In our experiments we explored the sensitivity of algorithm behavior, in particular word-error-rate and execution-time statistics, as a function of the settings of these parameters. In this section we present results that summarize what we learned. We report results for the $[1057, 244]$ LDPC code. We note that, in contrast to the simulation results for the AWGN channel presented in the last subsection, in this section we report on simulation results for the binary symmetric channel (BSC). We assume the BSC results from hard-decision demodulation of a BPSK ± 1 sequence transmitted over an AWGN channel. The resulting relation between the crossover probability p of the equivalent BSC-p and the SNR of the AWGN channel is $p = \mathrm{Q}\left(\sqrt{2R \cdot 10^{SNR/10}}\right)$, where R is the rate of the code and $\mathrm{Q}(\cdot)$ is the Q-function. We reported on WER performance of ADMM decoding for this code and channel in [3].

We first explore the effects of the choice of ϵ and μ on the error rate. We comment that as long as $t_{\max} > 300$ the choice of $t_{\max}$ does not significantly affect the WER. In Fig. 5 we plot WER as a function of the number of bits of stopping tolerance, i.e., $-\log_2(\epsilon)$. In Fig. 6 we plot WER as a function of μ. Each data point is based on more than 200 decoding errors.

From these two figures we conclude that the performance of the ADMM decoder depends only weakly on the settings of these two parameters, as long as the parameters are chosen sufficiently large. For instance $\epsilon \geq 10^{-4}$ and $\mu \geq 2$ should do. This means that the implementer has great latitude in the choice of these parameters and can make, e.g., hardware-compatible choices. Furthermore, the results on ending tolerance give hints as to the needed precision of the algorithm. If algorithmic precision is on the order of the needed ending tolerance we expect to observe similar error rates.
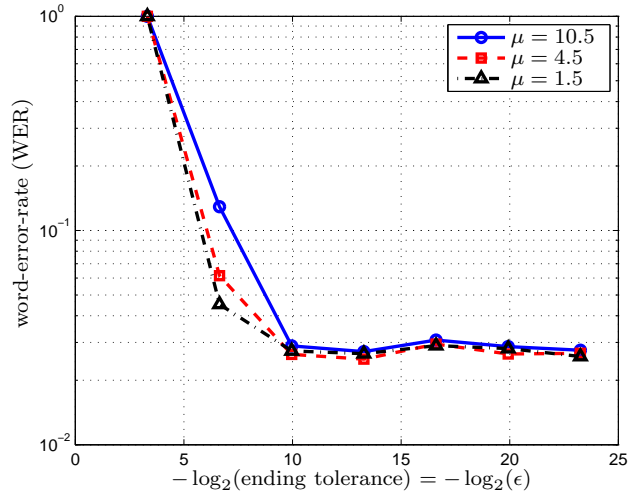
Figure 5: Word error rate (WER) of the $[1057, 244]$ LDPC code for the BSC plotted as a function of error tolerance ϵ for three difference penalty parameters μ. The SNR simulated is $5dB$. The maximum number of iterations t_{max} is set equal to 250.
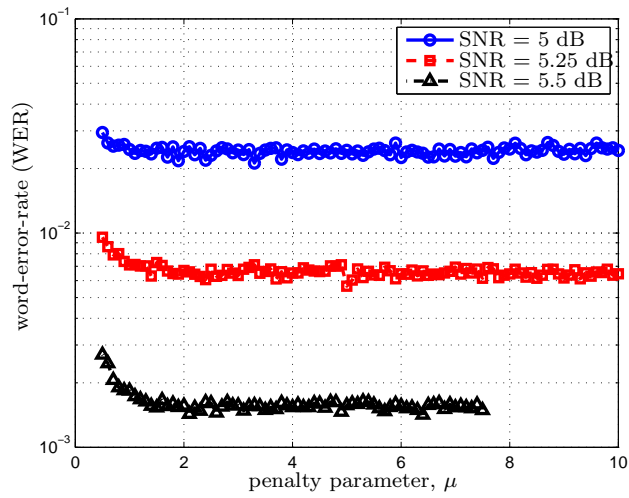

Figure 6: Word error rate (WER) of the $[1057, 244]$ LDPC code for the BSC plotted as a function of penalty parameter μ. Error tolerance $\epsilon = 10^{-4}$, and maximum number of iterations $t_{max} = 500$.
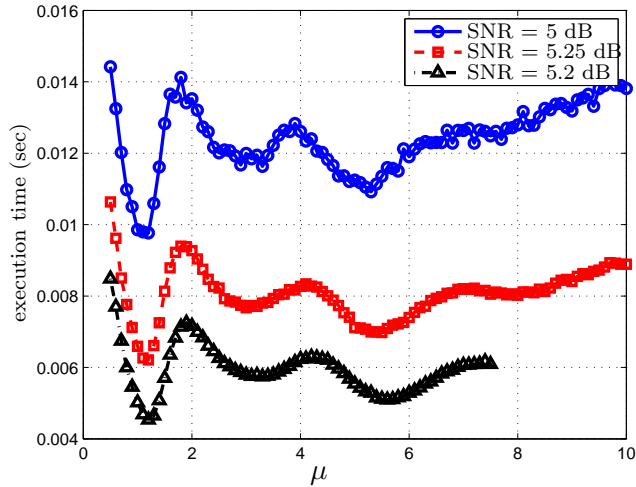
Figure 7: Average execution time (in seconds) of ADMM decoding the $[1057, 244]$ code simulated over the BSC plotted as a function of penalty parameter μ for three distinct SNRs.

We next study the effect of parameter section on average decoding time. All time statistics were collected on a 2GHz Intel(R) Xeon(R) CPU. In Fig. 7 we plot average decoding time as a function of μ for three SNRs. For all three ending tolerance is fixed at $\epsilon = 10^{-4}$. Note that based on Fig. 6 we should choose $\mu > 1.5$ for best WER performance. We see some weak variability in average decoding time as a function of the choice of μ.

Now, understanding the various parameters we can tune, we summarize the choices made for our simulation results presented in Sec. 5.1. For all simulations we made the following choices: (i) error tolerance $\epsilon = 10^{-5}$, (ii) penalty $\mu = 5$, (iii) maximum number of iterations $t_{\max} = 600$ for the $[2540, 1320]$ code and $t_{\max} = 500$ for the $[1057, 244]$ code.

Overrelaxation A significant improvement in average decoding time results from implementing an "over-relaxed" version of ADMM. Over-relaxed ADMM is discussed in [5, section 3.4.3] as a method for improving convergence speed while retaining convergence guarantees.

The over-relaxation parameter γ must be in the range $1 \leq \gamma < 2$. If $\gamma \geq 2$ convergence guarantees are lost. We did simulated $\gamma > 2$ and observed an increase in average decoding time. In Fig. 8 we plot the effect on average decoding time of over-relaxed versions of the ADMM decoder for $1 \leq \gamma \leq 1.9$. These plots are for the length-2640 Margulis code simulated over the AWGN channel at an SNR of 2.8dB. We observe that the average decoding time drops by a factor of about 50% over the range of γ. The improvement is roughly the same for the set of penalty parameters studied, $\mu \in \{1.5, 2, 2.5, 3\}$. The take-away is that by choosing over-relaxation parameter $\gamma = 1.9$ we can double decoding efficiency without degradation in error-rate.

While we did not use overrelaxation in the previously discussed experiments, we would encourage interested developers to explore proper settings of γ in their implementations.
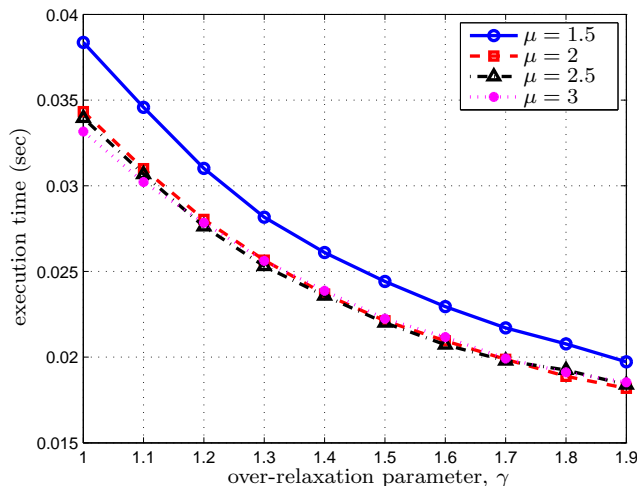
Figure 8: Average execution time for ADMM decoding the $[2640, 1320]$ Margulis code simulated over the AWGN channel at SNR = 2.8dB. Execution time (in seconds) is plotted as a function of over-relaxation parameter γ for four different penalty parameters $\mu \in \{1.5, 2, 2.5, 3\}$.

6 Conclusion

In this paper we apply the ADMM template to the LP decoding problem introduced in [15]. A main technical hurdle was the development of an efficient method of projecting a point onto the parity polytope. We accomplished this in two steps. We first introduced a new "two-slice" representation of points in the parity polytope. We then used the representation to show that the projection via an efficient waterfilling-type algorithm. We demonstrate the effectiveness of our decoding technique on two codes, on the rate-0.5 $[2640, 1320]$ "Margulis" LDPC code and the rate-0.77 $[1057, 244]$ LDPC code studied in [55]. We find that while similar in many aspects there are some significant difference between the decoding behavior of LP and sum-product BP decoding. On one hand, the waterfall of LP decoding initiates at slightly higher SNR than that of sum-product BP decoding. But, on the other, LP decoding does not seem to have an error floor. Fully understanding LP decoding performance in this high-SNR regime is an important future direction. What we have seen is that LP decoding, when implemented in a distributed, scalable manner using ADMM is a strong competitor to BP in the high-SNR regime. It allows LP decoding to be applied to long block-length codes, to be implemented as a message-passing algorithm using a very simple message update schedule, and to execute as fast as BP.

Acknowledgements

References

[1] M. V. Afonso, J. M. Bioucas-Dias, and M. A. T. Figueiredo. An augmented Lagrangian approach to the constrained optimization formulation of imaging inverse problems. *IEEE Transactions on Image Processing*, 20(3):681–695, 2011.

[2] A. Arora, D. Steuer, and C. Daskalakis. Message-passing algorithms and improved LP decoding. In *ACM Symposium on Theory of Computing (STOC)*, May 2009.

[3] S. Barman, X. Liu, S. C. Draper, and B. H. Recht. Decomposition methods for large-scale linear-programming decoding. In *Proc. Allerton Conf. on Communication, Control and Computing*, Monticello, IL, Sept. 2011.

[4] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, Belmont, MA, 2003.

[5] 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.

[6] D. Burshtein. Linear complexity approximate LP decoding of LDPC codes: generalizations and improvements. In *Proc. Int. Symp. Turbo Codes and Related Topics*, Lausanne, Switzerland, Sept. 2008.

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

[8] B. K. Butler and P. H. Siegel. Error floor approximation for LDPC codes in the AWGN channel. In *Proc. Allerton Conf. on Communication, Control and Computing*, Monticello, IL, Sept. 2011.

[9] B. K. Butler and P. H. Siegel. Error floor approximation for LDPC codes in the AWGN channel. *Arxiv preprint 1202.2826*, 2012.

[10] C. Daskalakis, A. G. Dimakis, R. M. Karp, and M. J. Wainwright. Probabilistic analysis of linear programming decoding. *IEEE Trans. Inform. Theory*, 54(8):3365–3578, Aug 2008.

[11] L. Dolecek, P. Lee, Z. Zhang, V. Anatharam, B. Nikolic, and M. J. Wainwright. Predicting error floors of structured LDPC codes: deterministic bounds and estimates. *IEEE J. Select. Areas Commun.*, 27(6):908–917, Aug. 2009.

[12] S. C. Draper, J. S. Yedidia, and Y. Wang. ML decoding via mixed-integer adaptive linear programming decoding. In *Proc. Int. Symp. Inform. Theory*, Nice, France, July 2007.

[13] J. Feldman. *Decoding Error-Correcting Codes via Linear Programming*. PhD thesis, Mass. Instit. of Tech., 2003.

[14] J. Feldman, T.Malkin, R. A. Servedio, C. Stein, and M. J. Wainwright. LP decoding corrects a constant fraction of errors. In *Proc. Int. Symp. Inform. Theory*, Chicago, IL, June 2004.

[15] J. Feldman, M. J. Wainwright, and D. Karger. Using linear programming to decoding binary linear codes. *IEEE Trans. Inform. Theory*, 51:954–972, Mar. 2005.

[16] G. D. Forney. Codes on graphs: normal realizations. *IEEE Trans. Inform. Theory*, 47(2):520–548, Feb. 2001.

[17] M. P. C. Fossorier. Quasicyclic, low-density parity-check codes from circulant permutation matrices. *IEEE Trans. Inform. Theory*, 50(8):1788–1793, Aug. 2004.

[18] B. J. Frey, R. Koetter, and A. Vardy. Signal-space characterization of iterative decoding. *IEEE Trans. Inform. Theory*, 47:766–781, Feb. 2001.

[19] R. W. Hamming. Error detecting and error correcting codes. *Bell Syst. Tech. J.*, 29(2):147–160, 1950.

[20] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold. Regular and irregular progressive edge-growth Tanner graphs. *IEEE Trans. Inform. Theory*, pages 386–398, Jan. 2005.

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

[22] R. Koetter and P. O. Vontobel. Graph-covers and iterative decoding of finite length codes. In *Proc. Int. Symp. Turbo Codes and Related Topics*, Brest, France, 2003.

[23] F. R. Kschischang, B. J. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47:498–519, Feb. 2001.

[24] H. Liu, W. Qu, B. Liu, and J. Chen. On the decomposition method for linear programming decoding of LDPC codes. *IEEE Trans. Commun.*, 58(12):3448–3458, Dec. 2010.

[25] H.-A. Loeliger. An introduction to factor graphs. *IEEE Signal Proc. Mag.*, 21(1):28 – 41, Jan. 2004.

[26] D. J. C. MacKay. Encyclopedia of sparse graph codes. Available at http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[27] D. J. C. MacKay and M. S. Postol. Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes. *Electronic Notes in Theoretical Computer Science*, 74(0):97–104, 2003.

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

[29] A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing. An augmented Lagrangian approach to constrained MAP inference. In *Proceedings of the International Conference on Machine Learning*, 2011.

[30] O. Milenkovic, D. Leyba, and N. Kashyap. Shortened array codes of large girth. *IEEE Trans. Inform. Theory*, 52(8):3707–3722, Aug. 2006.

[31] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[32] M. E. O'Sullivan. Algebraic constructions of sparse matrices with large girth. *IEEE Trans. Inform. Theory*, 52(2):718–727, feb 2006.

[33] T. Richardson. Error floors of LDPC codes. In *Proc. Allerton Conf. on Communication, Control and Computing*, Monticello, IL, Oct. 2003.

[34] T. Ruozzi, J. Thaler, and S. Tatikonda. Graph covers and quadratic minimization. In *Proc. Allerton Conf. on Communication, Control and Computing*, Monticello, IL, Oct. 2009.

[35] W. Ryan and S. Lin. *Channel Codes: Classical and Modern*. Cambridge University Press, 2009.

[36] M.-H. N. Taghavi, A. Shokrollahi, and P. H. Siegel. Efficient implementation of linear programming decoding. *IEEE Trans. Inform. Theory*, 55(9):5960–5982, Sept. 2010.

[37] M.-H. N. Taghavi and P. H. Siegel. Adaptive methods for linear programming decoding. *IEEE Trans. Inform. Theory*, 54(12):5396–5410, Dec. 2008.

[38] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, 27:533–547, Sept. 1981.

[39] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel. Construction of irregular LDPC codes with low error floors. In *Proc. Int. Conf. Commun.*, pages 3125–3129, Anchorage, AK, May 2003.

[40] P. Vontobel. Interior-point algorithms for linear-programming decoding. In *UCSD Workshop Inform. Theory Apps.*, San Diego, CA, Jan. 2008.

[41] P. O. Vontobel. Counting in graph covers: A combinatorial characterization of the Bethe entropy function. *ArXiv e-prints*, Nov. 2010.

[42] P. O. Vontobel and R. Koetter. On the relationship between linear programming decoding and min-sum algorithm decodign. In *IEEE Int. Symp. Inform. Theory and Apps.*, Parma, Italy, Oct. 2004.

[43] P. O. Vontobel and R. Koetter. Towards low-complexity linear-programming decoding. In *Proc. Int. Symp. Turbo Codes and Related Topics*, Munich, Germany, Apr. 2006.

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

[45] T. Wadayama. Interior point decoding for linear vector channels based on convex optimization. In *Proc. Int. Symp. Inform. Theory*, pages 1493–1497, Toronto, CA, July 2008.

[46] T. Wadayama. An LP decoding algorithm based on primal path-following interior point method. In *Proc. Int. Symp. Inform. Theory*, pages 389–393, Seoul, Korea, July 2009.

[47] Y. Wang, S. C. Draper, and J. S. Yedidia. Hierarchical and high-girth QC LDPC codes. *archive preprint 1111.0711*, 2011. submitted to *IEEE Trans. Inform. Theory*.

[48] Y. Wang and M. Fossorier. Doubly generalized LDPC codes. In *Proc. Int. Symp. Inform. Theory*, pages 669–673, Seattle, WA, July 2006.

[49] Y. Wang, J. S. Yedidia, and S. C. Draper. Multi-stage decoding of LDPC codes. In *Proc. Int. Symp. Inform. Theory*, Seoul, South Korea, July 2009.

[50] K. Yang, J. Feldman, and X. Wang. Nonlinear programming approaches to decoding low-density parity-check codes. *IEEE J. Select. Areas Commun.*, 24:1603–1613, Aug. 2006.

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

[52] J. Yedidia, W. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Trans. Inform. Theory*, 51(7):2282 – 2312, July 2005.

[53] J. S. Yedidia. The alternating direction method of multipliers as a message-passing algorithms. Talk delivered at the Princeton Workshop on Counting, Inference and Optimization, Oct 2011.

[54] J. S. Yedidia. A taxonomy suggested by Jonathan Yedidia in personal correspondence, Jan 2012.

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

[56] J. Zhang, J. S. Yedidia, and M. P. C. Fossorier. Low-latency decoding of EG LDPC codes. *Journal of Lightwave Technology*, 25:2879–2886, Sept. 2007.

[57] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. J. Wainwright. Design of LDPC decoder for improved low error rate performance: Quantization and algorithm choices. *IEEE Trans. Commun.*, 57(11):3258–3268, Nov. 2009.

A Dual Subgradient Ascent

If we work with an (un-augmented) Lagrangian

$$L_0(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) := \boldsymbol{\gamma}^T \boldsymbol{x} + \sum_{j \in \mathcal{J}} \boldsymbol{\lambda}_j^T (\boldsymbol{P}_j \boldsymbol{x} - \boldsymbol{z}_j)$$

the dual subgradient ascent method consists of the iterations:

$$\boldsymbol{x}^{k+1} := \operatorname{argmin}_{\boldsymbol{x} \in \mathcal{X}} L_0(\boldsymbol{x}, \boldsymbol{z}^k, \boldsymbol{\lambda}^k)$$
$$\boldsymbol{z}^{k+1} := \operatorname{argmin}_{\boldsymbol{z} \in \mathcal{Z}} L_0(\boldsymbol{x}^k, \boldsymbol{z}, \boldsymbol{\lambda}^k)$$
$$\boldsymbol{\lambda}_j^{k+1} := \boldsymbol{\lambda}_j^k + \mu \left(\boldsymbol{P}_j \boldsymbol{x}^{k+1} - \boldsymbol{z}_j^{k+1} \right).$$

Note here that the $\boldsymbol{x}$ and $\boldsymbol{z}$ updates are run with respect to the k iterates of the other variables, and can be run completely in parallel.

The $\boldsymbol{x}$-update corresponds to solving the very simple LP:

$$
\begin{array}{ll}
\text{minimize} & \left(\boldsymbol{\gamma} + \sum_{j \in \mathcal{J}} \boldsymbol{P}_j^T \boldsymbol{\lambda}_j^k \right)^T \boldsymbol{x} \\
\text{subject to} & \boldsymbol{x} \in [0, 1]^N.
\end{array}
$$

This results in the assignment:

$$\boldsymbol{x}^{k+1} = \theta \left(-\boldsymbol{\gamma} - \sum_{j \in \mathcal{J}} \boldsymbol{P}_j^T \boldsymbol{\lambda}_j^k \right)$$

where

$$\theta(t) = \begin{cases} 1 & t > 0 \\ 0 & t \leq 0 \end{cases}$$

is the Heaviside function.

For the $\boldsymbol{z}$-update, we have to solve the following LP for each $j \in \mathcal{J}$:

$$
\begin{array}{ll}
\text{maximize} & \boldsymbol{\lambda}_j^{k^T} \boldsymbol{z}_j \\
\text{subject to} & \boldsymbol{z}_j \in \mathbb{PP}_d.
\end{array}
\tag{A.1}
$$

Maximizing a linear function over the parity polytope can be performed in linear time. First, note that the optimal solution necessarily occurs at a vertex, which is a binary vector with an even hamming weight. Let r be the number of positive components in the cost vector $\boldsymbol{\lambda}_j^k$. If r is even, the vector $\boldsymbol{v} \in \mathbb{PP}_d$ which is equal to 1 where $\boldsymbol{\lambda}_j^k$ is positive and zero elsewhere is a solution of (A.1), as making any additional components nonzero decreases the cost as does making any of the components equal to 1 smaller. If r is odd, we only need to compare the cost of the vector equal to 1 in the $r - 1$ largest components and zero elsewhere to the cost of the vector equal to 1 in the $r + 1$ largest components and equal to zero elsewhere.

The procedure to solve (A.1) is summarized in Algorithm 3. Note that finding the smallest positive element and largest nonnegative element can be done in linear time. Hence, the complexity of Algorithm 3 is $O(d)$.

While this subgradient ascent method is quite simple, it is requires vastly more iterations than the ADMM method, and thus we did not pursue this any further.

Algorithm 3 Given a binary d-dimensional vector $\boldsymbol{c}$, maximize $\boldsymbol{c}^T \boldsymbol{z}$ subject to $\boldsymbol{z} \in \mathbb{PP}_d$.

1: Let r be the number of positive elements in $\boldsymbol{c}$.
2: **if** r is even **then**
3:     **Return** $\boldsymbol{z}^*$ where $z_i^* = 1$ if $c_i > 0$ and $z_i^* = 0$ otherwise.
4: **else**
5:     Find index i_p of the smallest positive element of $\mathbf{c}$.
6:     Find index i_n of the largest non-positive element of $\mathbf{c}$.
7:     **if** $c_{i_p} > c_{i_n}$ **then**
8:         **Return** $\boldsymbol{z}^*$ where $\boldsymbol{z}_i^* = 1$ if $c_i > 0$, $z_{i_n}^* = 1$, and $z_i^* = 0$ otherwise.
9:     **else**
10:        **Return** $\boldsymbol{z}^*$ where $\boldsymbol{z}_i^* = 1$ if $c_i > 0$ and $i \neq i_p$, $z_{i_p}^* = 0$, and $z_i^* = 0$ for all other i.
11:     **end if**
12: **end if**