Decomposition Methods for Large Scale LP Decoding

Siddharth Barman, Xishuo Liu, Stark C. Draper, *Member, IEEE*, and Benjamin Recht, *Member, IEEE*

***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 error-correcting codes 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 direction method of multipliers (ADMM), to develop efficient distributed algorithms for LP decoding. The key enabling technical result is a "two-slice" characterization of the parity polytope, the polytope formed by taking the convex hull of all codewords of the single parity check code. This new characterization simplifies the representation of points in the polytope. Using this simplification, we develop an efficient algorithm for Euclidean norm projection onto the parity polytope. This projection is required by the ADMM decoder and its solution allows us to use LP decoding, with all its theoretical guarantees, to decode large-scale error correcting codes efficiently. We present numerical results for LDPC codes of lengths more than 1000. The waterfall region of LP decoding is seen to initiate at a slightly higher SNR than for sum-product BP, however an error floor is not observed for LP decoding, which is not the case for BP. Our implementation of LP decoding using the ADMM executes as fast 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.**

***Index Terms*—Alternating direction method of multipliers (ADMM), belief propagation (BP), decomposition methods, error floors, Euclidean projection, graphical models, iterative algorithms, linear programming (LP) decoding, low-density parity check (LDPC) codes, parity polytope, permutohedron.**

I. Introduction

WHILE the problem of error correction decoding dates back at least to Hamming's seminal work in the 1940s [1], the idea of drawing upon techniques of convex optimization to solve such problems apparently dates only to Feldman's 2003 Ph.D. thesis [2], [3]. 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 [4]–[7] demonstrates that the bit-error-rate performance of LP decoding is competitive with that of standard sum-product (and min-sum) belief propagation (BP) decoding. Furthermore, LP decoding comes with a certificate of correctness (ML certificate) [3]—verifying with probability 1 when the decoder has found the ML codeword and, if a high-quality expander [8], [9] or high-girth [10] 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 LP. 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 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 weaknesses of arrangements in the graphical structure of the code (variously termed "pseudocodewords," "near-codewords," "trapping sets," "instantons," "absorbing sets" [11]–[15]), from the suboptimal BP decoding algorithm, and from the particulars of the implementation of BP. Two natural approaches to improve error floor performance are to design codes with fewer problematic arrangements [16]–[21] or to develop improved decoding algorithms. As LP decoders have been observed to have lower error rate at high SNRs compared to BP decoding [7], [22]–[24], 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

Manuscript received April 11, 2012; revised February 25, 2013; accepted April 29, 2013. Date of publication September 10, 2013; date of current version November 19, 2013. This work was supported in part by the National Science Foundation under Grants CCF-1217058 and CCF-1148243 and in part by the Office of Naval Research under Award N00014-13-1- 0129. The work in this paper was performed while the authors were affiliated with the University of Wisconsin, Madison. This paper was presented in part at the Allerton Conference on Communication, Control, and Computing, Monticello, IL, USA, 2011.

S. Barman is with the Center for the Mathematics of Information, California Institute of Technology, CA 91125 USA (e-mail: barman@caltech.edu).

X. Liu is with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706 USA (e-mail: xliu94@wisc.edu).

S. C. Draper is with the Department of Electrical and Computer Engineering, University of Toronto, ON M5S 3G4, Canada (e-mail: stark.draper@utoronto.ca).

B. Recht is with the Department of Computer Sciences, University of Wisconsin, Madison, WI 53706 USA (e-mail: brecht@cs.wisc.edu).

Communicated by D. Burshtein, Associate Editor for Coding Techniques.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2013.2281372

vertex corresponds to the ML codeword. When the optimum vertex is noninteger (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 [25], [26], [6]. In some settings, one can regularly attain ML performance with few additional constraints [22].

In this paper, we produce a fast decomposition algorithm based on the alternating direction method of multipliers [27] (ADMM). This is a classic technique in convex optimization and has gained a good deal of popularity lately for solving problems in compressed sensing [28] 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 resemblance to BP. Variable nodes update their estimates of their true values 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 straightforward. 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 IV, we demonstrate that this projection can be computed in linearithmic time in the degree of the check. This in turn enables us to develop LP decoders with computational complexity comparable to (and sometimes much faster than) BP decoders.

The structure of the decoding LP has been examined before in pursuit of efficient implementation. The first attempt was by Vontobel and Koetter [5], [30] where the authors used a coordinate-ascent method to develop distributed message-passing type algorithms to solve the LP. Their method requires scheduling updates cyclically on all edges in order to guarantee convergence. However, when their approach is matched with an appropriate message-passing schedule, as determined by Burshtein in [23] and [31], converge to the optimal solution can be attained with a computational complexity which scales linearly with the block length. Further, interior-point [32]–[35] and revised-simplex [36] approaches have also been applied. In a separate approach Yedida *et al.* in [22] 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 between iterations in ADMM. In Section II, we introduce the LP decoding problem and introduce notation. We set up the general formulation of ADMM problems in Section III and specialize the formulation to the LP decoding problem. In Section IV, we present our main technical contributions wherein we develop the efficient projection algorithm. We present numerical results in Section V and make some final remarks in Section VI.

II. 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 $i_1 < i_2 < \cdots < i_d$. 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

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

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, $\gamma_i := \log[W(\tilde{x}_i|0)/W(\tilde{x}_i|1)]$. 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]

[1]This derivation applies to all binary-input DMCs. In the simulations of Section V, 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 $\gamma_i = \log[W(0|0)/W(0|1)] = \log[(1-p)/p]$.

Feldman *et al.* [3] 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

$$\mathbb{PP}_d = \text{conv}(\mathbb{P}_d) = \text{conv}(\{\boldsymbol{e} \in \{0,1\}^d \mid \|\boldsymbol{e}\|_1 \text{ is even}\}). \quad (2)$$

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

$$\text{minimize } \boldsymbol{\gamma}^T\boldsymbol{x} \text{ subject to } \boldsymbol{P}_j\boldsymbol{x} \in \mathbb{PP}_d \; \forall \, j \in \mathcal{J}. \quad (3)$$

The statement of the optimization problem in (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 [37], 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 [38], 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 [3] 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, $\sum_{\text{even}s}^{d} \mu_s = 1$. 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 parity polytope (see [38] and [3] for a proof)

$$\begin{aligned}
&0 \leq u_i \leq 1 \; \forall \; i \in [d] \\
&0 \leq z_{i,s} \leq \mu_s \; \forall \; i \in [d] \\
&\sum_{\text{even } s}^{d} \mu_s = 1 \\
&u_i = \sum_{\text{even } s}^{d} z_{i,s} \; \forall \; i \in [d] \\
&\sum_{i=1}^{d} z_{i,s} = s\mu_s \; \forall \; s \text{ even}, \; s \leq d.
\end{aligned}$$

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 IV-B, 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 toward 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.

III. Decoupled Relaxation and Optimization Algorithms

In this section, we present the ADMM formulation of the LP decoding problem and summarize our contributions. In Section III-A, we introduce the general ADMM template. We specialize the template to our problem in Section III-B. We state the algorithm in Section III-C and frame it in the language of message-passing in Section III-D.

A. ADMM Formulation

To make the LP (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

$$\begin{aligned}
&\text{minimize} \quad \boldsymbol{\gamma}^T\boldsymbol{x} \\
&\text{subject to } \boldsymbol{P}_j\boldsymbol{x} = \boldsymbol{z}_j \; \forall \; j \in \mathcal{J} \\
&\qquad \boldsymbol{z}_j \in \mathbb{PP}_d \; \forall \; j \in \mathcal{J} \\
&\qquad \boldsymbol{x} \in [0,1]^N.
\end{aligned} \quad (4)$$

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

$$\begin{aligned}
L_\mu(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) :=& \boldsymbol{\gamma}^T\boldsymbol{x} + \sum_{j \in \mathcal{J}} \boldsymbol{\lambda}_j^T(\boldsymbol{P}_j\boldsymbol{x} - \boldsymbol{z}_j) \\
&+ \frac{\mu}{2} \sum_{j \in \mathcal{J}} \|\boldsymbol{P}_j\boldsymbol{x} - \boldsymbol{z}_j\|_2^2.
\end{aligned} \quad (5)$$

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 [39], 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 [40]. For the interested reader, we provide a discussion of the standard dual ascent method in 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 $\boldsymbol{z}_1 \times \boldsymbol{z}_2 \times \cdots \times \boldsymbol{z}_{|\mathcal{J}|} \in \mathbb{PP}_d \times \mathbb{PP}_d \times \cdots \times \mathbb{PP}_d$, the $|\mathcal{J}|$-fold product of $\mathbb{PP}_d$. Then, we can succinctly write the iterations of ADMM as

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

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.

B. 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 (5), 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 or equal to 1, the components less than 0 or 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 nonzero 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.

Componentwise, the update rule corresponds to taking the average of the corresponding replica values $\boldsymbol{z}_j$ adjusted by 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

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

Each variable update can be done in parallel.

The $\boldsymbol{z}$-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 (5) 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

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

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

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

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

C. ADMM Decoding Algorithm

The complete ADMM-based LP decoding algorithm is specified in the Algorithm 1 box. We declare convergence when the following two conditions are satisfied: 1) replicas differ from the $\boldsymbol{x}$ variables by less than some tolerance $\epsilon > 0$, and 2) the value of each replica differs from its value in the previous iteration by less than ϵ.

Algorithm 1 Given an N-dimensional vector $\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}^N$, $M \times N$ parity check matrix $\boldsymbol{H}$, and parameters μ and ϵ, solve the decoding LP specified in (4)

1: Construct the negative log-likelihood vector $\boldsymbol{\gamma}$ based on received word $\tilde{\boldsymbol{x}}$. Construct the $d \times N$ matrix $\boldsymbol{P}_j$ for all $j \in \mathcal{J}$.
2: Initialize $\boldsymbol{z}_j$ and $\boldsymbol{\lambda}_j$ as the all zeros vector for all $j \in \mathcal{J}$. Initialize iterate $k = 0$. For simplicity, we only specify iterate k when determining stopping criteria.
3: **repeat**
4: **for all** $i \in \mathcal{I}$ **do**
5: Update $x_i \leftarrow \prod_{[0,1]}\Big(\frac{1}{|\mathcal{N}_v(i)|}\Big(\sum_{j \in \mathcal{N}_v(i)}\Big(z_j^{(i)} - \frac{1}{\mu}\lambda_j^{(i)}\Big) - \frac{1}{\mu}\gamma_i\Big)\Big)$.
6: **end for**
7: **for all** $j \in \mathcal{J}$ **do**
8: Set $\boldsymbol{v}_j = \boldsymbol{P}_j\boldsymbol{x} + \boldsymbol{\lambda}_j/\mu$.
9: Update $\boldsymbol{z}_j \leftarrow \Pi_{\mathbb{PP}_d}(\boldsymbol{v}_j)$ where $\Pi_{\mathbb{PP}_d}(\cdot)$ means project onto the parity polytope.
10: Update $\boldsymbol{\lambda}_j \leftarrow \boldsymbol{\lambda}_j + \mu(\boldsymbol{P}_j\boldsymbol{x} - \boldsymbol{z}_j)$.
11: **end for**
12: $k \leftarrow k + 1$.
13: until $\sum_j \|\boldsymbol{P}_j\boldsymbol{x}^k - \boldsymbol{z}_j^k\|_2^2 < \epsilon^2 Md$ and $\sum_j \|\boldsymbol{z}_j^k - \boldsymbol{z}_j^{k-1}\|_2^2 < \epsilon^2 Md$
return $\boldsymbol{x}$.

A convergence analysis for ADMM is provided in [27]. We base the following analysis on [41, Th. 1]. The output of ADMM decoding $\hat{\boldsymbol{x}}$ satisfies the order relation

$$\boldsymbol{\gamma}^T\hat{\boldsymbol{x}} - \boldsymbol{\gamma}^T\boldsymbol{x}^* = O\left(\frac{Md\mu}{T}\right),$$

where we recall that $\boldsymbol{\gamma}$ is the cost vector, M is the number of checks, d is the check dimension, μ is the ADMM penalty parameter, and where T and $\boldsymbol{x}^*$ are, respectively, the number of iterations and the solution to the LP decoding problem. Since for LDPC codes $M = O(N)$,

$$\boldsymbol{\gamma}^T\hat{\boldsymbol{x}} - \boldsymbol{\gamma}^T\boldsymbol{x}^* = O\left(\frac{N}{T}\right).$$

This means that for a given $\delta > 0$, ADMM outputs vector $\hat{\boldsymbol{x}}$ with $O(1)$ iterations, such that $\boldsymbol{\gamma}^T\hat{\boldsymbol{x}} - \boldsymbol{\gamma}^T\boldsymbol{x}^* < N\delta$. For each iteration, the $\boldsymbol{x}$-update has $O(N)$ complexity, the $\boldsymbol{z}$-update has $O(M)$ complexity, and the $\boldsymbol{\lambda}$-update has $O(M)$ complexity.

Combining the above results, the complexity of ADMM decoding is given by the following proposition.

Proposition 1: Let $\boldsymbol{x}^*$ be a solution of the LP decoding problem. For any $\delta > 0$, Algorithm 1 will, in $O(N)$ time, determine a vector $\hat{\boldsymbol{x}}$ that satisfies the following bound:

$$\boldsymbol{\gamma}^T\hat{\boldsymbol{x}} - \boldsymbol{\gamma}^T\boldsymbol{x}^* < N\delta.$$

We note that the experiments we present in Section V demonstrate that this convergence estimate is frequently conservative; we often see convergence in a dozen iterations or fewer. (See Fig. 6, for experiments regarding iteration requirements.)

D. ADMM Decoding as Message-Passing Algorithm

We now present a message-passing interpretation of the ADMM approach to LP decoding as presented in Algorithm 1. For simplicity, we establish this interpretation by identifying messages passed between variable nodes and check nodes on a Tanner graph.

We denote by $x_{ij}(k)$ the replica associated with the edge joining variable node $i \in \mathcal{I}$ and check 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 variable node i to check 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 update 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.

Another nice way to see the decoupling of the decoding problem, and the connection to message passing, is to use the normal factor graph formalism [42]. Then, in a manner similar to that taken in [22, Sec. III-A], replicas are associated with edges of the normal graph and one creates a dynamics of replicas, alternately trying to satisfy equality and parity-check constraints. The dynamics of ADMM are, in general, distinct from those of the "Divide-and-Conquer" algorithm studied in [22] but related, as recently shown in [43].

IV. 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 IV-A by describing the structured geometry of $\mathbb{PP}_d$ and laying out the results that will follow in ensuing sections. In Section IV-B, we prove a few necessary lemmas illustrating some of the symmetry structure of the parity polytope. In Section IV-C, 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 IV-D, we present the projection algorithm.

A. 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 $\mathbb{PP}_d$.

1) 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 $\mathbb{PP}_d$:

$$\mathbb{P}_d^r = \{\mathbf{e} \in \{0,1\}^d \mid \|\boldsymbol{e}\|_1 = r\}, \tag{6}$$

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

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

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

$$\mathbb{PP}_d^r = \text{conv}(\mathbb{P}_d^r) = \text{conv}(\{\boldsymbol{e}\in\{0,1\}^d \mid \|\boldsymbol{e}\|_1 = r\}). \tag{7}$$

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,

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

3) Third, define the affine hyperplane consisting of all vectors whose components sum to r as

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

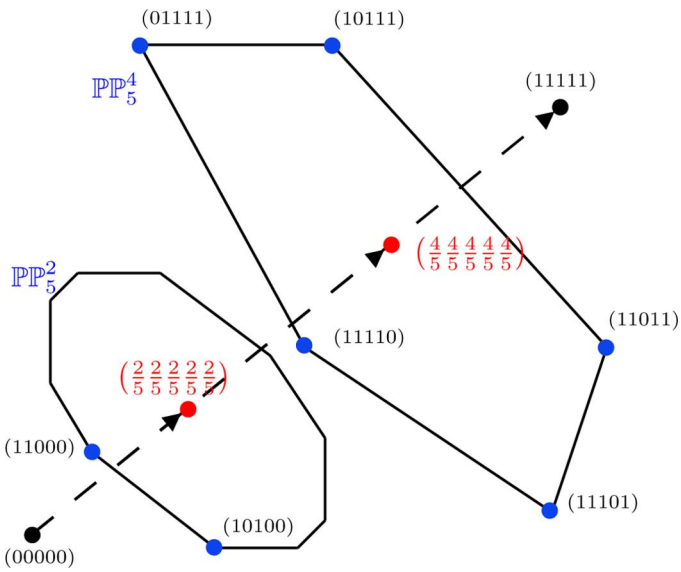

Fig. 1. Parity polytope $\mathbb{PP}_d$ can be expressed as the convex hull of "slices" through $\mathbb{PP}_d$, each of which contains all weight-r vertices. These sets, $\mathbb{PP}_d^r$ are permutahedra. They are all orthogonal to the line segment connecting the origin to the all-ones vector. The geometry is sketched for $d = 5$.

where $\mathbf{1}$ is the length-d all-ones vector. We can visualize $\mathbb{PP}_d^r$ as a "slice" through 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 (7) is

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

for r an even integer.

4) 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 Section IV-B, we draw on a theorem from [44] 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 1) 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 point above, a convex combination of points in slices of different weights r. In other words, $\mathbf{u} = \sum_{0 \le r \le d\,:\, r \text{ even}} \alpha_r \mathbf{u}_r$ 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 2, that shows that two slices always suffices. In other words, we can always write $\boldsymbol{u} = \alpha \boldsymbol{u}_r + (1-\alpha)\boldsymbol{u}_{r+2}$ where $\boldsymbol{u}_r \in \mathbb{PP}_d^r$, $\boldsymbol{u}_{r+2} \in \mathbb{PP}_d^{r+2}$, $0 \le \alpha \le 1$, and $r = \lfloor \|\boldsymbol{u}\| \rfloor_{\text{even}}$, 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 Section IV-C, 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 $\Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ be 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 Section IV-D, we develop our projection algorithm. In short, 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 [45], we showed that, given α, this projection can be accomplished in two steps. We first scale $\mathbb{PP}_d^r$ by the convex weighting parameter α to obtain $\alpha\mathbb{PP}_d^r = \{\mathbf{x} \in \mathbb{R}^d | 0 \le x_i \le \alpha, \sum_{i=1}^d x_i = \alpha r\}$ and project $\boldsymbol{v}$ onto $\alpha\mathbb{PP}_d^r$. 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 a 1-D line search (using, for example, the secant method [46, p. 188]) to determine the optimal value for α and thence the desired projection.

In contrast to the original approach, in Section IV-D, 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 [45].

B. 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.

Proposition 2: $\boldsymbol{u} \in \mathbb{PP}_d$ if and only if $\boldsymbol{\Sigma u}$ is in the parity polytope for every $d \times d$ 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

$$\text{minimize}_{\boldsymbol{z}} \|\boldsymbol{v} - \boldsymbol{z}\|_2 \text{ subject to } \boldsymbol{z} \in \mathbb{PP}_d\,. \tag{8}$$

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 3: 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 to 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 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 1: 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 [44] and references therein).

Theorem 1: 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 1, $\boldsymbol{u} \in [0,1]^d$ is in $\mathbb{PP}_d^s$ if and only if

$$\sum_{k=1}^{q} u_k \leq \min(q, s) \quad \forall\, 1 \leq q < d, \tag{9}$$
$$\sum_{k=1}^{d} u_k = s. \tag{10}$$

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 1: A sorted vector $\boldsymbol{u} \in \mathbb{PP}_d$ if and only if there exist nonnegative coefficients $\{\mu_s\}_{\text{even} s \leq d}$ such that

$$\sum_{s \text{ even}}^{d} \mu_s = 1, \quad \mu_s \geq 0. \tag{11}$$
$$\sum_{k=1}^{q} u_k \leq \sum_{s \text{ even}}^{d} \mu_s \min(q, s) \quad \forall\, 1 \leq q < d \tag{12}$$
$$\sum_{k=1}^{d} u_k = \sum_{s \text{ even}}^{d} \mu_s s. \tag{13}$$

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 (11)–(13). Conversely, if $\boldsymbol{u}$ satisfies (11)–(13), 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 1 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 nonzero μ_s.

C. 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 $r \leq a$. 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.

Lemma 2 ("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 1 tells us that there is always (at least one) set $\{\mu_s\}$ that satisfy (11)–(13). 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 $\|\boldsymbol{u}\|_1 = \alpha r + (1 - \alpha)(r + 2)$:

$$\alpha = \frac{2 + r - \|\boldsymbol{u}\|_1}{2}. \tag{14}$$

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 (11)–(13) 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, (11) and (13) are both satisfied. Further, for the candidate set the relations (12) and (13) simplify to

$$\sum_{k=1}^{q} u_k \leq \alpha \min(q, r) + (1 - \alpha) \min(q, r + 2), \ \forall\, 1 \leq q < d, \tag{15}$$
$$\sum_{k=1}^{d} u_k = \alpha r + (1 - \alpha)(r + 2). \tag{16}$$

To show that (15) is satisfied is straightforward for the cases $q \leq r$ and $q \geq r+2$. First consider any $q \leq r$. Since $\min(q,r) = \min(q,r+2) = q$, $u_k \leq 1$ for all k, and there are only q terms, (15) must hold. Second, consider any $q \geq r+2$. We use (16) to write $\sum_{k=1}^{q} u_k = \alpha r + (1-\alpha)(r+2) - \sum_{q+1}^{d} u_k$. Since $u_k \geq 0$ this is upper bounded by $\alpha r + (1-\alpha)(r+2)$ which we recognize as the right-hand side of (15) since $r = \min(q,r)$ and $r+2 = \min(q,r+2)$.

It remains to verify only one more inequality in (15) namely the case when $q = r+1$, which is

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

To show that the above inequality holds, we maximize the right-hand side of (12) 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 details are as follows: since $\boldsymbol{u} \in \mathbb{PP}_d$ any valid choice for $\{\mu_s\}$ must satisfy (11) which, for $q = r+1$, is

$$\sum_{k=1}^{r+1} u_k \leq \sum_{s \text{ even}}^{d} \mu_s \min(s, r+1). \tag{17}$$

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 LP

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

The first two constraints are simply (11) and (13). Recognizing $\alpha r + (1-\alpha)(r+2) = r + 2 - 2\alpha$, the dual program is

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

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 (17) 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 (14) satisfies the requirements of Lemma 1 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 $\text{conv}(\mathbb{PP}_d^r \cup \mathbb{PP}_d^{r+2}) \subseteq \mathbb{PP}_d$. ■

A useful consequence of Theorem 1 is the following corollary.

Corollary 1: 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 1, $\boldsymbol{u} \in \mathbb{PP}_d^s$ 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 3: 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,

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

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 exists 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 componentwise 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 1 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 $[\rho_L, \rho_U]$ ■

D. 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.

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

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., $\boldsymbol{v} = \boldsymbol{Q}\boldsymbol{u}$.
Compute $\hat{\boldsymbol{z}} \leftarrow \Pi_{[0,1]^d}(\boldsymbol{v})$.
Assign $r = \lfloor \|\hat{\boldsymbol{z}}\|_1 \rfloor_{\text{even}}$
if $r = d$ **then**
  return $\boldsymbol{z}^* = \hat{\boldsymbol{z}}$.
end if
if $r \leq d-2$ **then**
  $\beta_{\max} = \frac{1}{2}[v_{r+1} - v_{r+2}]$.
else
  $\beta_{\max} = v_{r+1}$.
end if
Define $\boldsymbol{f}_r$ as in (22).
if $\boldsymbol{f}_r^T\hat{\boldsymbol{z}} \leq r$ **then**
  return $\boldsymbol{z}^* = \hat{\boldsymbol{z}}$.
end if
Assign $\mathcal{E}_1 = \{v_i - 1 \mid 1 \leq i \leq r+1\}$,
  $\mathcal{E}_2 = \{-v_i \mid r+2 \leq i \leq d\}$.
Construct the set of breakpoints
  $\mathcal{B} := \{\beta \in \mathcal{E}_1 \cup \mathcal{E}_2 \mid 0 \leq \beta \leq \beta_{\max}\}$ by
  merging the sorted points in $\mathcal{E}_1$ and $\mathcal{E}_2$ so that entries in $\mathcal{B}$ satisfy $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_{|\mathcal{B}|}$.
Initialize a as the smallest index such that $0 < \hat{z}_a < 1$.
Initialize b as the largest index such that $0 < \hat{z}_b < 1$.
Initialize sum $V = \boldsymbol{f}_r^T\hat{\boldsymbol{z}}$.
for $i = 1$ to $|\mathcal{B}|$ **do**
  Set $\beta_0 \leftarrow \beta_i$.
  if $\beta_i \in \mathcal{E}_1$ **then**
    Update $a \leftarrow a - 1$.
    Update $V \leftarrow V + v_a$.
  else
    Update $b \leftarrow b + 1$
    Update $V \leftarrow V - v_b$.
  end if
  if $i < |\mathcal{B}|$ and $\beta_i \neq \beta_{i+1}$ **then**
    $\Lambda \leftarrow (a-1) + V - \beta_0(b-a+1)$
    if $\Lambda \leq r$ **then break**
  else if $i = |\mathcal{B}|$ **then**
    $\Lambda \leftarrow (a-1) + V - \beta_0(b-a+1)$
  end if
end for
if $\Lambda > r$ **then**
  Compute $\beta_{\text{opt}} \leftarrow \beta_0 - \frac{r-\Lambda}{b-a+1}$.
else
  $\beta_0 \leftarrow \beta_{i-1}$
  $a \leftarrow |\{j \mid v_j - \beta_0 > 1\}|$
  $b \leftarrow r + 2 + |\{j \mid v_j + \beta_0 0\}|$
  $V \leftarrow \sum_{j=a-1}^{r+1} v_j - \sum_{j=r+2}^{b+1} v_j$
  $\beta_{\text{opt}} \leftarrow \frac{V+b-r}{b-a+1}$
end if
return $\boldsymbol{z}^* = \boldsymbol{Q}^T\Pi_{[0,1]^d}(\boldsymbol{v} - \beta_{\text{opt}}\boldsymbol{f}_r)$.

Given a vector $\boldsymbol{v} \in \mathbb{R}^d$, set $r = \lfloor \|\Pi_{[0,1]^d}(\boldsymbol{v})\|_1 \rfloor_{\text{even}}$. From Lemma 3, we know that the constituent parity of $\boldsymbol{z}^* := \Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ is r. We also know that if $\boldsymbol{v}$ is sorted in descending order then $\boldsymbol{z}^*$ will also be sorted in descending order. 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 2 implies that $\boldsymbol{z}^*$ is a convex combination of vectors of Hamming weight r and $r+2$. Using inequality (15), we get that a d-vector $\boldsymbol{z} \in [0,1]^d$, with

$$\sum_{i=1}^{d} z_i = \alpha r + (1-\alpha)(r+2), \tag{18}$$

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

$$\sum_{k=1}^{q} z_{(k)} \leq \alpha \min(q,r) + (1-\alpha)\min(q, r+2)\ \forall\ 1 \leq q < d, \tag{19}$$

where $z_{(k)}$ denotes the kth largest component of $\boldsymbol{z}$. As we saw in the proof of Lemma 1, the fact that the components of $\boldsymbol{z}$ are no more than one implies that inequalities (19) are satisfied for all $q \leq r$. Also, (18) enforces the inequalities for $q \geq r+2$. Therefore, inequalities in (19) for $q \leq r$ and $q \geq r+2$ are redundant. Note that in addition we can eliminate the variable α by solving (18) giving $\alpha = 1 + \frac{r - \sum_{k=1}^{d} z_k}{2}$ [see also (14)]. Therefore, for a sorted vector $\boldsymbol{v}$, we can write the projection onto $\mathbb{PP}_d$ as the optimization problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{v} - \boldsymbol{z}\|_2^2 \\ \text{subject to} \quad & 0 \leq z_i \leq 1\ \forall\ i \\ & \boldsymbol{S}\boldsymbol{z} \geq 0 \\ & 0 \leq 1 + \frac{r - \sum_{k=1}^{d} z_k}{2} \leq 1 \qquad (20) \\ & \sum_{k=1}^{r+1} z_k \leq r - \frac{r - \sum_{k=1}^{d} z_k}{2}. \qquad (21) \end{aligned}$$

The last two constraints can be simplified as follows: first, constraint (20) simplifies to $r \leq \sum_{k=1}^{d} z_k \leq r+2$. Next, define the vector

$$\boldsymbol{f}_r = (\underbrace{1, 1, \ldots, 1}_{r+1}, \underbrace{-1, -1, \ldots, -1}_{d-r-1})^T \tag{22}$$

we can rewrite inequality (21) as $\boldsymbol{f}_r^T \boldsymbol{z} \leq r$. Using these simplifications yields the final form of our quadratic program.

Problem PQP:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{v}-\boldsymbol{z}\|_2^2 \\ \text{subject to} \quad & 0 \leq z_i \leq 1 \; \forall \; i \qquad (23) \\ & \boldsymbol{S}\boldsymbol{z} \geq \boldsymbol{0} \qquad (24) \\ & r \leq \boldsymbol{1}^T \boldsymbol{z} \leq r+2 \qquad (25) \\ & \boldsymbol{f}_r^T \boldsymbol{z} \leq r. \qquad (26) \end{aligned}$$

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, $\boldsymbol{\nu}$, $\boldsymbol{\eta}$, ξ, $\boldsymbol{\theta}$, and ζ:

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

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

$$\begin{aligned} \boldsymbol{z} = \boldsymbol{v} - \beta\boldsymbol{f}_r - \boldsymbol{\nu} + \boldsymbol{\eta} - (\xi-\zeta)\boldsymbol{1} + \boldsymbol{S}^T\boldsymbol{\theta}. \qquad (27) \\ 0 \leq \beta \;\perp\; \boldsymbol{f}_r^T\boldsymbol{z} - r \leq 0 \\ \boldsymbol{0} \leq \boldsymbol{\nu} \;\perp\; \boldsymbol{z} \leq \boldsymbol{1} \\ \boldsymbol{0} \leq \boldsymbol{\eta} \;\perp\; \boldsymbol{z} \geq \boldsymbol{0} \\ \boldsymbol{0} \leq \boldsymbol{\theta} \;\perp\; \boldsymbol{S}\boldsymbol{z} \geq \boldsymbol{0} \\ 0 \leq \xi \;\perp\; \boldsymbol{1}^T\boldsymbol{z} - r - 2 \leq 0 \\ 0 \leq \zeta \;\perp\; \boldsymbol{1}^T\boldsymbol{z} - r \geq 0. \end{aligned}$$

A vector $\boldsymbol{z}$ that satisfies (27) 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

$$\boldsymbol{z}(\beta) := \Pi_{[0,1]^d}(\boldsymbol{v} - \beta\boldsymbol{f}_r)\,. \qquad (28)$$

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

Lemma 4: There exists a $\beta_{\text{opt}} \in [0, \beta_{\max}]$ such that $\boldsymbol{z}(\beta_{\text{opt}})$ 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, $z(\beta)$ 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 1. 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:

$$\begin{aligned} \frac{\partial \boldsymbol{f}_r^T\boldsymbol{z}(\beta)}{\partial\beta} &= \boldsymbol{f}_r^T\frac{\partial\boldsymbol{z}(\beta)}{\partial\beta} \\ &= \sum_{k:0<z_k(\beta)<1} -f_{r,k}^2 \qquad (29) \\ &= -\left|\{k \mid 1 \leq k \leq d, 0 < z_k(\beta) < 1\}\right| \\ &< 0 \end{aligned}$$

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

First note that $\boldsymbol{z}(\beta_{\text{opt}})$ is feasible for problem PQP. We need only to verify (25). Recalling that r is defined as $r = \lfloor\|\Pi_{[0,1]^d}(\boldsymbol{v})\|_1\rfloor_{\text{even}}$, we get the lower bound

$$\boldsymbol{1}^T\boldsymbol{z}(\beta_{\text{opt}}) \geq \boldsymbol{f}_r^T\boldsymbol{z}(\beta_{\text{opt}}) = r.$$

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

To complete the proof, we need only find dual variables to certify the optimality. Setting ξ, ζ, and $\boldsymbol{\theta}$ to zero, and $\boldsymbol{\nu}$ and $\boldsymbol{\eta}$ to the values required to satisfy (27) provides the necessary assignments to satisfy the KKT conditions. ■

Lemma 4 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

$$\mathcal{A}(\beta) := \{k \mid 1 \leq k \leq d, 0 < z_k(\beta) < 1\}\,. \qquad (30)$$

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

$$\mathcal{C}(\beta) := \{k \mid 1 \leq k \leq d, z_k(\beta) = 1\}\,. \qquad (31)$$

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

$$\mathcal{Z}(\beta) := \{k \mid 1 \leq k \leq d, z_k(\beta) = 0\}\,. \qquad (32)$$

Note that with these definitions, we have

$$\begin{aligned} \boldsymbol{f}_r^T\boldsymbol{z}(\beta) &= |\mathcal{C}(\beta)| + \sum_{j\in\mathcal{A}(\beta)}(z_j - \beta) \\ &= |\mathcal{C}(\beta)| - \beta|\mathcal{A}(\beta)| + \sum_{j\in\mathcal{A}(\beta)} z_j. \qquad (33) \end{aligned}$$

Our algorithm simply increases β 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 (33).

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:

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

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

$$\mathcal{B} := \{\beta \in \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{L}_1 \cup \mathcal{L}_2 | \, 0 \le \beta \le \beta_{\max}\} \cup \{0, \beta_{\max}\}.$$

The following proposition reduces the search space by identifying unnecessary breakpoints.

Proposition 4: $\beta_{\text{opt}} \le v_{r+1}$ and $\beta_{\text{opt}} < 1 - v_{r+2}$.

Proof: If $v_{r+1} > -v_{r+2}$, then $v_{r+1} > \frac{1}{2}(v_{r+1} - v_{r+2}) = \beta_{\max}$. Therefore $\beta_{\text{opt}} < \beta_{\max} < v_{r+1}$. If $v_{r+1} \le -v_{r+2}$, then for all $\hat{\beta} \ge v_{r+1}$, we have $\boldsymbol{f}_r^T z(\hat{\beta}) \le r$. This means that for all $\hat{\beta} > v_{r+1}$, $\boldsymbol{f}_r^T z(\hat{\beta}) < r$ and hence $\hat{\beta}$ cannot be β_{opt}.

In addition, $\beta_{\text{opt}} < 1 - v_{r+2}$. This is because for all $\hat{\beta} \ge 1 - r_{r+2}$, we have $\|z(\hat{\beta})\|_1 \ge r + 2$. Therefore, $z(\hat{\beta})$ cannot be the projection. ∎

The proposition above indicates that the points in $\mathcal{L}_1$ and $\mathcal{L}_2$ are not necessary when identifying β_{opt}. The reason is that the only possible solution in these two sets is $\beta_{\text{opt}} = v_{r+1} \in \mathcal{L}_1$. However, since v_{r+1} is the smallest breakpoint, we can directly solve for β_{opt} by inspecting the second smallest breakpoint and use (33). This procedure is captured in Algorithm 2. We use $\mathcal{B} = \left\{\beta \in \cup_{j=1}^{2} \mathcal{E}_j \mid 0 \le \beta \le \beta_{\max}\right\}$ as the set of breakpoints in Algorithm 2. This set contains at most d points.

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 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 (33). 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 one sort (sorting the input vector) and inspections of at most d breakpoints. Thus, the total complexity of the algorithm is linear plus the time for the sort, which is $O(d \log d)$. We make two remarks on the computational complexity of Algorithm 2. First, d is small for LDPC codes. Thus, the asymptotic complexity is less important. The complexity from the sorting operation is negligible. Second, we can consider Algorithm 2 as the check node operation. Compared with an exact sum-product BP check node update, where hyperbolic tangents and logarithms are used, our algorithm uses only basic arithmetic operations and thus is more hardware friendly. We demonstrate in numerical results that the actual run time for each iteration of ADMM decoding is similar to that for BP decoding.

V. Numerical Results and Implementation

In this section, we present simulation results for the ADMM decoder and discuss various aspects of our implementation. In Section V-A, we present word-error-rate (WER) results for two particular LDPC codes as well as for an ensemble of random $(3, 6)$-regular LDPC codes. We note that numerical results for the $[155, 64]$ Tanner code [47] is reported in our previous work [45]. In Section V-B, we discuss how the various parameters choices in ADMM affect decoding performance, as measured by error rate and by decoding time.

A. Performance Comparisons Between ADMM and BP Decoding

In this section, we present simulation results of ADMM decoding and compare to sum-product BP decoding. The parameters used for ADMM are as follows: 1) error tolerance $\epsilon = 10^{-5}$, 2) penalty $\mu = 3$, 3) maximum number of iterations $t_{\max} = 1000$, and 4) over-relaxation parameter (cf. Section V-B) $\rho = 1.9$. The maximum number of iterations for BP decoding is also 1000. We discuss parameter choices in detail in Section V-B.

We first present results for two particular codes over the additive white Gaussian noise (AWGN) channel with binary inputs. The first code is the $[2640, 1320]$ rate-0.5, $(3, 6)$-regular Margulis LDPC code [48]. The second is a $[1057, 813]$ rate-0.77, $(3, 13)$-regular LDPC code obtained from [49]. This code is also studied by Yedidia *et al.* [22]. We choose both codes as they have been chosen in the past to study error floor performance. Then, we present results for an ensemble of 100 randomly generated $(3, 6)$-regular LDPC codes of length 1002 similar to [24]. We simulate the binary symmetric channel (BSC) and compare ADMM with BP decoding in three aspects: error rate, number of iterations, and execution time.[2]

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., [13], [48], [50]). Recently it has been noted [50], [51] 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 implementation of "nonsaturating" sum-product BP, which follows the implementation of [50], [51], 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 two highest SNRs (2.8 and 3 dB), for which we collected 130 and 32 errors, respectively. For nonsaturating BP decoding, we collect 43 and 13 respective errors at $\text{SNR} = 2.6$, 2.7 dB.

The first aspect to note is that while the LP decoder has a waterfall, the waterfall initiates at a slightly higher SNR (about 0.4 dB higher 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 [7] and [22]. It is

[2]We note that results for ADMM decoding of the $[155, 64]$ Tanner code are given in [45], results that match those given in [26].

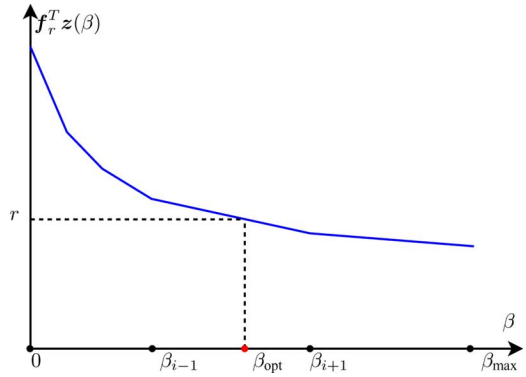

Fig. 2. Since there are a finite number of breakpoints (at most $2d+2$) and the function $\boldsymbol{f}_r^T \boldsymbol{z}(\beta)$ is linear between breakpoints, we can solve for β_{opt} in linear time. See (22) and (28) for definitions of $\boldsymbol{f}_r$ and $\boldsymbol{z}(\beta)$, respectively.

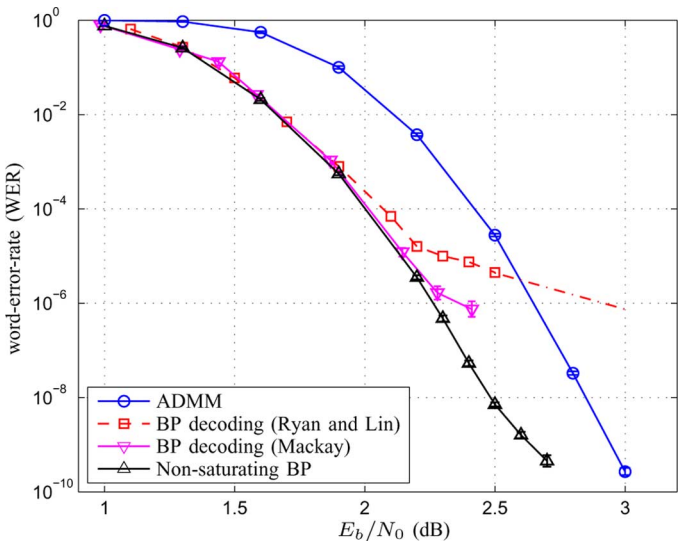

Fig. 3. Word error rate (WER) of the $[2640, 1320]$ "Margulis" LDPC code used on the AWGN channel plotted as a function of SNR. The WER performance of ADMM is compared to that of nonsaturating sum-product BP, as well as to results for (saturating) sum-product BP from Ryan and Lin [48] and from MacKay and Postol [13].

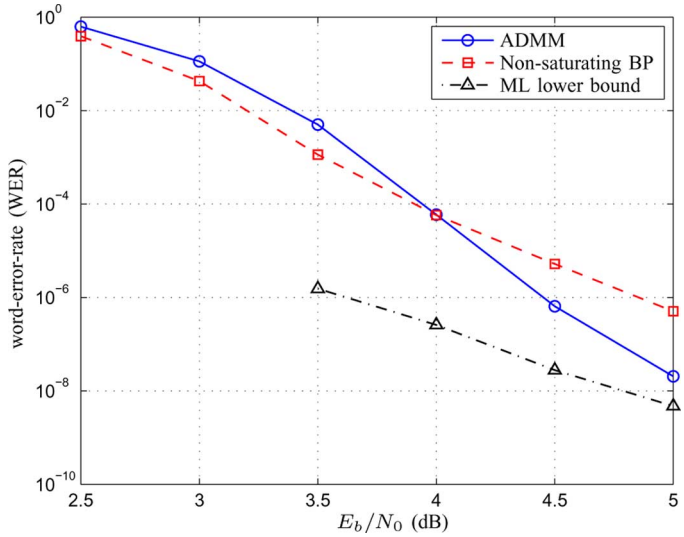

Fig. 4. Word error rate (WER) of the $[1057, 813]$ LDPC code used on the AWGN channel plotted as a function of SNR. The WER performance of ADMM is compared to that of nonsaturating sum-product BP, as well as to an estimated lower-bound on ML decoding.

worth mentioning that expressing BP decoding as optimization over the Bethe free energy is introduced in [52]. Further studies such as [53] show that BP and LP decoding, when expressed using the Bethe free energy, are different in the objective function. Therefore, 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 ADMM decoding at WERs above 10^{-10}. When decoding of this code using the nonsaturating version of sum-product, we observe a weak error floor at WERs near 10^{-9}, in which regime the waterfall of ADMM is continuing to steepen. In this regime, we found that the nonsaturating BP decoder is oscillating, as discussed in [54] and [55]. We note that we have not simulated WERs at 10^{-10} or lower due to the limitation of our computational resources. It would be extremely interesting to see the performance of ADMM decoding at WERs lower than 10^{-10}.

Fig. 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 $\text{SNR} = 5$ dB, 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 noninteger 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 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). Similar 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.

In Figs. 5–7, we present comparisons between ADMM decoding and BP decoding using an ensemble of 100 randomly generated $(3, 6)$-regular LDPC codes of length 1002. We eliminated codes that have parallel edges, thus all codes have girth of at least four. However, cycles of length four or greater are not eliminated. We will use this ensemble to understand the error performance and the computational performance of LP and of BP decoding. For this study, we simulate the BSC in order to match the settings used in [24]. All data points presented are averaged across the 100 codes in the ensemble. For each code, we collect more than five word-errors.

In Fig. 5, we plot the average word-error-rate (WER) and bit-error-rate (BER) observed for both BP and ADMM decoding. We observe similar comparisons between ADMM and BP decoding found in previous examples. In particular, note the error

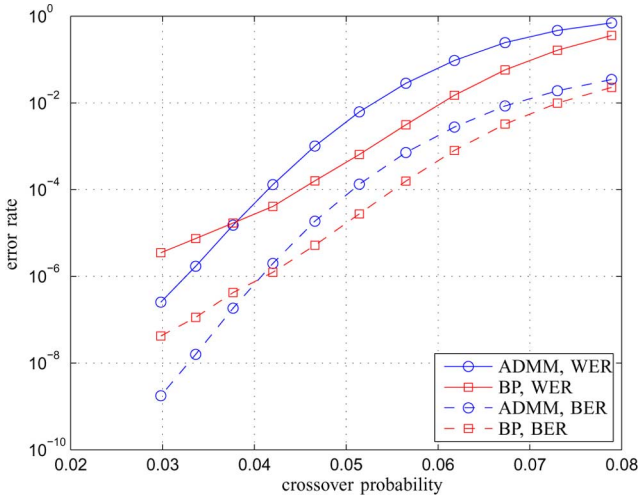

Fig. 5. Word error rate (WER) and bit-error-rate (BER) of the $(3, 6)$-regular random LDPC code used on the BSC plotted as a function of crossover probability. The error rate performance of ADMM is compared to that of saturating sum-product BP. Results are averaged over 100 randomly generated codes.

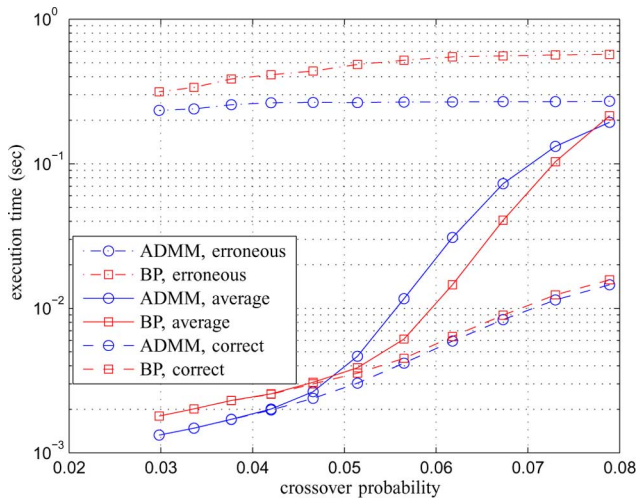

Fig. 7. Execution time of the $(3, 6)$-regular random LDPC code used on the BSC plotted as a function of crossover probability. The execution time of ADMM is compared to that of saturating sum-product BP. Results are averaged over 100 randomly generated codes.

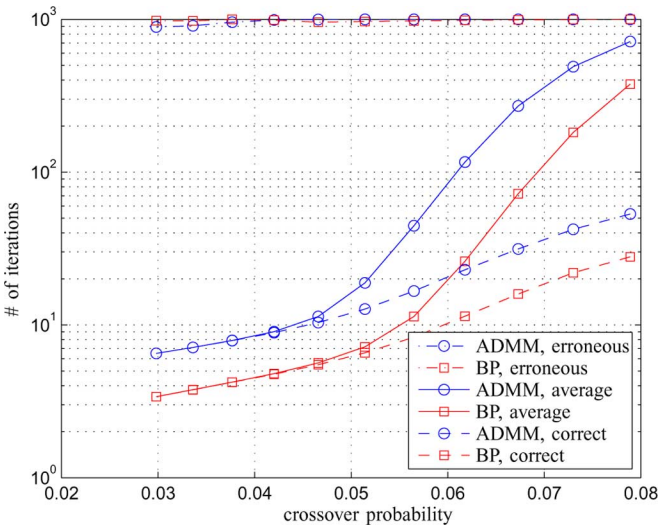

Fig. 6. Number of iterations of the $(3, 6)$-regular random LDPC code used on the BSC plotted as a function of crossover probability. The number of iterations of ADMM is compared to that of saturating sum-product BP. Results are averaged over 100 randomly generated codes.

floor flare observable in BP at crossover probabilities of about 0.045 and below. No such flare is evident in ADMM.

In Fig. 6, we plot a comparison of the iteration requirements of ADMM and BP decoding for the same ensemble of codes. We plot three curves for each decoder: the average number of iterations required to decode, the average number of iterations required to decode when decoding is correct, and the average number required when decoding is erroneous. We observe that ADMM decoding needs more iterations to decode than BP does. However, the gap between the decoders is roughly constant (on this log scale) meaning the ratio of iterations required is roughly constant. Thus, the trend for increased iterations at higher crossovers is the same for both decoders. Further, both decoder reach the maximum number of allowable iterations when errors occur. An important observations is that although we allow up to 1000 iterations in our simulations, the average number of iterations required by ADMM for correct decoding events is quite small at *all* SNRs. This means that ADMM converges quickly to a correct codeword, but more slowly to a pseudocodeword. We discuss further the effect of choice of the maximum number of iterations in Section V-B.

In Fig. 7, we plot the time comparisons between ADMM and BP decoding using the same methodology. For this figure, we plot results for the *saturating* version of BP where we have tried to optimized our implementations. This decoder executes *much* more quickly than our implementation of nonsaturating BP. Both decoders are simulated on the same CPU configurations. We make two observations. First, when measured in terms of execution time, the computational complexity of ADMM and BP are similar. This observation holds for all crossover probabilities simulated. Second, ADMM decoding is faster than BP when decoding is correct. Combining these results with those on iteration count from Fig. 6, we conclude that the execution time for each iteration of ADMM is shorter than for BP.

Given the importance of error floor effects in high reliability applications, and the 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 also argued in [50] and [51]) is that numerical precision effects can dramatically affect code performance in the high-SNR regime. From a practical point of view, a real-world implementation would use fixed precision arithmetic. Thus, understanding the behavior of ADMM decoding under finite precision is extremely important.

A second point made by comparing these codes is that the performance of an algorithm, e.g., nonsaturating BP, can vary dramatically from code to code (Margulis versus 1057), and the performance of a code can vary dramatically from algorithm to algorithm (BP versus ADMM). For each algorithm, we might think about three types of codes [56]. 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., [57], [58], [16], [59], [18], 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. Some advantageous numerical properties of ADMM are as follows: first, ADMM has rigorous convergence guarantees [27]. Second, ADMM has historically be observed to be quite robust to parameter choices and precision settings [27]. This robustness will be further demonstrated in Section V-B. Third, 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). Due to these numerical properties, we expect that the ADMM decoder will be a strong competitor to BP in applications that demand ultrahigh reliabilities.

B. 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 Margulis LDPC code as used in the AWGN channel. This is consistent with the simulations presented in the last section.

We first explore the effects of the choice of ϵ and μ on error rate. We comment that as long as $t_{\max} > 300$ the choice of $t_{\max}$ does not significantly affect the WER. This effect is also evidenced in Fig. 6 where the average number of iterations required in correct decoding events is seen to be small. In Fig. 8, we plot WER as a function of the number of bits of stopping tolerance, i.e., $-\log_2(\epsilon)$. In Fig. 9, 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 error performance of the ADMM decoder depends weakly only on the settings of these two parameters. A sufficiently large ϵ parameter and a moderate μ parameter are good enough to achieve the desired error rate. For instance $\epsilon \geq 10^{-4}$ and $\mu \geq 1$ should do. This means that the design engineer 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.

We next study the effect of parameter section, on average decoding time. All time statistics were collected on a 3 GHz Intel(R) Core(TM) 2 CPU. In Fig. 10, we plot average decoding time as a function of μ for three SNRs. For all three the ending tolerance is fixed at $\epsilon = 10^{-5}$. We see some variability in average decoding time as a function of the choice of μ. Recalling

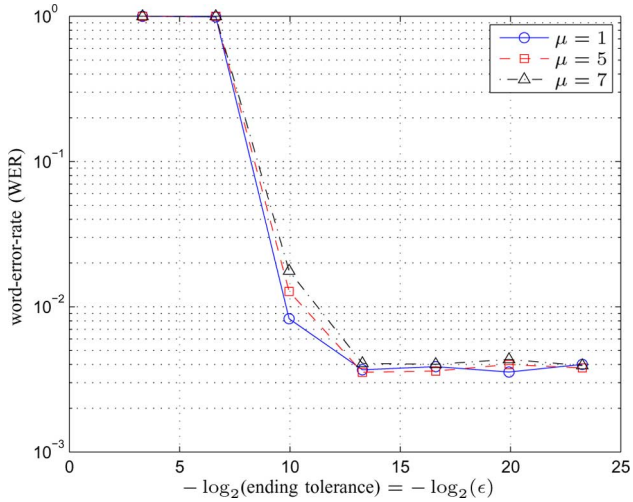

Fig. 8. Effect of the error tolerance ϵ on word error rate (WER). The WER of the Margulis LDPC code for the AWGN channel plotted as a function of error tolerance ϵ for three difference penalty parameters μ. The SNR simulated is 2.2 dB. The maximum number of iterations $t_{\max}$ is set equal to 1000.

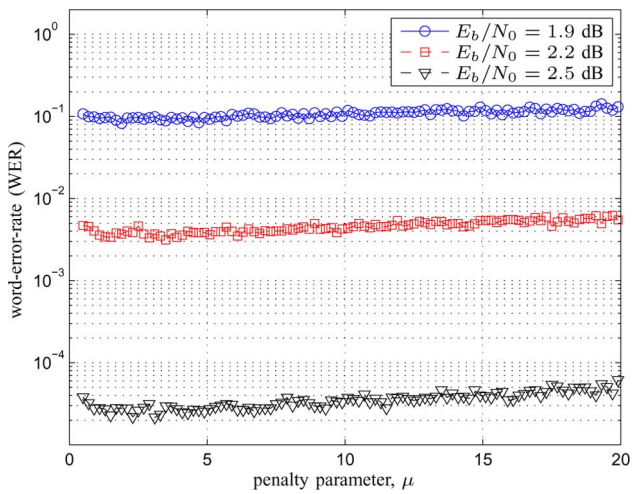

Fig. 9. Effect of the penalty parameter μ on word error rate (WER). The WER of the Margulis LDPC code for the AWGN channel plotted as a function of penalty parameter μ. Error tolerance $\epsilon = 10^{-5}$, and maximum number of iterations $t_{\max} = 1000$.

from Fig. 9 that we should choose $\mu \in [1, 10]$ for good WER performance, we conclude that $\mu \in [2, 5]$ is a good choice in term of both error- and time-performance.

Overrelaxation: A significant improvement in average decoding time results from implementing an "over-relaxed" version of ADMM. Overrelaxed ADMM is discussed in [27, Sec. 3.4.3] as a method for improving convergence speed while retaining convergence guarantees.

The overrelaxation parameter ρ must be in the range $1 \leq \rho < 2$. If $\rho \geq 2$ convergence guarantees are lost. We did simulated $\rho > 2$ and observed an increase in average decoding time. In Fig. 11, we plot the effect on average decoding time of overrelaxed versions of the ADMM decoder for $1 \leq \rho \leq 1.9$. These plots are for the Margulis code simulated over the AWGN channel at an SNR of 2.8 dB. We observe that the average decoding time drops by a factor of about 50% over the range of ρ. The improvement is roughly constant across the set of penalty

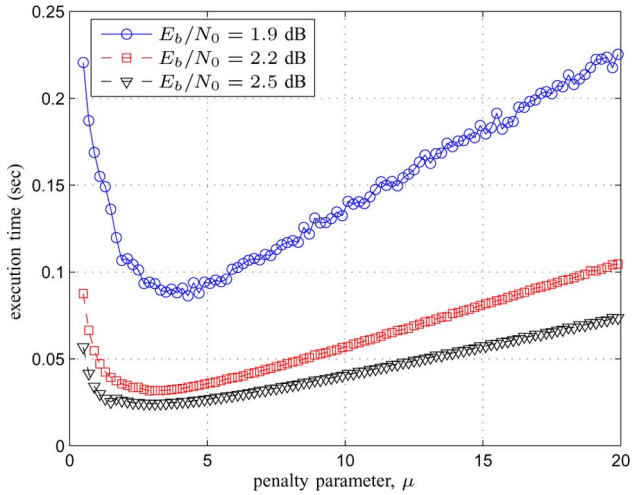

Fig. 10. Effect of the penalty parameter μ on execution time. Average execution time (in seconds) of ADMM decoding the Margulis code simulated over the AWGN channel plotted as a function of penalty parameter μ for three distinct SNRs.

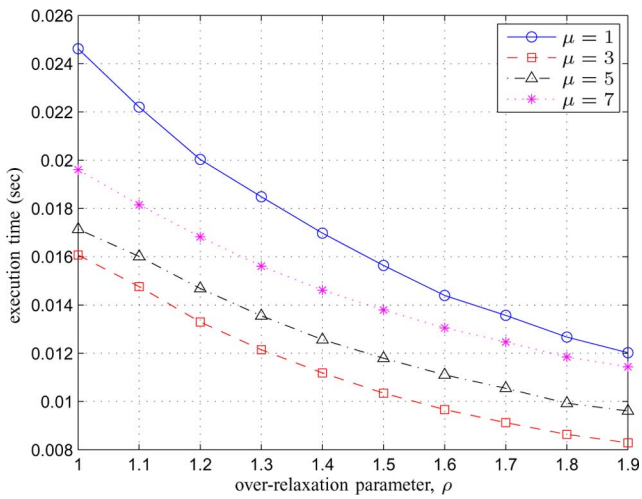

Fig. 11. Effect of the overrelaxation parameter ρ on execution time. Average execution time for ADMM decoding the Margulis code simulated over the AWGN channel at $E_b/N_0 = 2.8$ dB. Execution time (in seconds) is plotted as a function of overrelaxation parameter ρ for four different penalty parameters $\mu \in \{1, 3, 5, 7\}$.

parameters studied: $\mu \in \{1, 3, 5, 7\}$. By choosing the overrelaxation parameter $\rho = 1.9$, we can double decoding efficiency without degradation in error-rate.

While we did not use overrelaxation in the experiments on parameter choices reported in Figs. 8–10, we would encourage interested readers to explore proper settings of ρ in their implementations.

VI. Conclusion

In this paper, we apply the ADMM template to the LP decoding problem introduced in [3]. 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 attain the projection via an efficient waterfilling-type algorithm.

We demonstrate the effectiveness of our decoding technique on the rate-0.5 $[2640, 1320]$ "Margulis" LDPC code, the rate-0.77 $[1057, 813]$ LDPC code studied in [22], and an ensemble of randomly generated $(3, 6)$-regular LDPC codes. We find that while the decoding behaviors of LP and sum-product BP are similar in many aspects there are also significant differences. On one hand, the waterfall of LP decoding initiates at slightly higher SNR than that of sum-product BP decoding. On the other, LP decoding does not seem to have an error floor. 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 implemented as a message-passing algorithm with a simple update schedule that can be applied to long block-length codes with execution times similar to BP.

An immediate question is how to close the SNR gap that exists between LP and BP decoding at low SNRs. In recent work [60], the authors introduce a penalized LP that increases the relative cost of pseudocodewords vis-a-vis all-integer solutions. While the resulting optimization is (slightly) nonconvex the ADMM framework introduced herein can be applied with negligible (or no) increase in computational complexity. This slight modification closes the SNR gap between BP and LP decoding while retaining the high-SNR behavior of LP decoding. Fully understanding the performance of the modified LP decoders introduced in [60] is an important future direction. Other interesting directions include application of the decoder to other classes of codes, study of finite-precision effects, and generalization of the two-slice representation to a larger class of easy-to-project upon polytopes.

Appendix

We note that ADMM is not the only method to decompose LP decoding problem. Here we present another decomposition method using an (unaugmented) Lagrangian. This method is not as efficient as the ADMM decomposition. We hope to share this algorithm for readers interested in developing other decomposition methods.

First we construct an unaugmented 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

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

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

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

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

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 $j \in \mathcal{J}$:

$$\begin{aligned} &\text{maximize} && {\boldsymbol{\lambda}_j^k}^T \boldsymbol{z}_j \\ &\text{subject to} && \boldsymbol{z}_j \in \mathbb{PP}_d. \end{aligned} \tag{34}$$

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 (34), 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 (34) 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 $O(d)$.

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**

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

ACKNOWLEDGMENT

The authors would like to thank M. Anderson, B. Butler, E. Bach, A. Dimakis, P. Siegel, E. Telatar, Y. Wang, J. Yedidia, and D. Zelený for useful discussions and references. The authors would also like to note that some of the simulation results presented herein would not have been possible without the resources and the computing assistance of the University of Wisconsin (UW), Madison, Center For High Throughput Computing (CHTC) in the Department of Computer Sciences. The CHTC is supported by UW-Madison and the Wisconsin Alumni Research Foundation, and is an active member of the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science.

REFERENCES

[1] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, no. 2, pp. 147–160, 1950.
[2] J. Feldman, "Decoding error-correcting codes via linear programming," Ph.D. dissertation, MIT, , MA, USA, 2003.
[3] J. Feldman, M. J. Wainwright, and D. Karger, "Using linear programming to decoding binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.
[4] P. O. Vontobel and R. Koetter, "On the relationship between linear programming decoding and min-sum algorithm decoding," presented at the IEEE Int. Symp. Inf. Theory and Appl., Parma, Italy, Oct. 2004.
[5] P. O. Vontobel and R. Koetter, "On low-complexity linear-programming decoding of LDPC codes," *Eur. Trans. Telecommun.*, vol. 18, no. 5, pp. 509–517, 2007.
[6] M.-H. N. Taghavi and P. H. Siegel, "Adaptive methods for linear programming decoding," *IEEE Trans. Inf. Theory*, vol. 54, no. 12, pp. 5396–5410, Dec. 2008.
[7] Y. Wang, J. S. Yedidia, and S. C. Draper, "Multi-stage decoding of LDPC codes," presented at the Int. Symp. Inf. Theory, Seoul, Korea, Jul. 2009.
[8] J. Feldman, T. Malkin, R. A. Servedio, C. Stein, and M. J. Wainwright, "LP decoding corrects a constant fraction of errors," presented at the Int. Symp. Inf. Theory, Chicago, IL, USA, Jun. 2004.
[9] C. Daskalakis, A. G. Dimakis, R. M. Karp, and M. J. Wainwright, "Probabilistic analysis of linear programming decoding," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3365–3578, Aug. 2008.
[10] S. Arora, C. Daskalakis, and D. Steuer, "Message-passing algorithms and improved LP decoding," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, May 2009, pp. 3–12.
[11] B. J. Frey, R. Koetter, and A. Vardy, "Signal-space characterization of iterative decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 766–781, Feb. 2001.
[12] R. Koetter and P. O. Vontobel, "Graph-covers and iterative decoding of finite length codes," presented at the Int. Symp. Turbo Codes Related Topics, Brest, France, 2003.
[13] D. J. C. MacKay and M. S. Postol, "Weaknesses of Margulis and Ramanujan-Margulis low-density parity-check codes," *Electron. Notes Theor. Comput. Sci.*, vol. 74, no. 0, pp. 97–104, 2003.
[14] T. Richardson, "Error floors of LDPC codes," presented at the Allerton Conf. Commun., Control Comput., Monticello, IL, USA, Oct. 2003.
[15] 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. Sel. Areas Commun.*, vol. 27, no. 6, pp. 908–917, Aug. 2009.
[16] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.
[17] 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.*, Anchorage, AK, USA, May 2003, pp. 3125–3129.
[18] Y. Wang, S. C. Draper, and J. S. Yedidia, "Hierarchical and high-girth QC LDPC codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 7, pp. 4553–4583, Jul. 2013.
[19] J. Zhang, J. S. Yedidia, and M. P. C. Fossorier, "Low-latency decoding of EG LDPC codes," *J. Lightw. Technol.*, vol. 25, no. 9, pp. 2879–2886, Sep. 2007.
[20] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.
[21] Y. Wang and M. Fossorier, "Doubly generalized LDPC codes," in *Proc. Int. Symp. Inf. Theory*, Seattle, WA, USA, Jul. 2006, pp. 669–673.

[22] J. S. Yedidia, Y. Wang, and S. C. Draper, "Divide and concur and difference-map BP decoders for LDPC codes," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 786–802, Feb. 2011.
[23] D. Burshtein, "Iterative approximate linear programming decoding of LDPC codes with linear complexity," *IEEE Trans. Inf. Theory*, vol. 55, no. 11, pp. 4835–4859, Nov. 2009.
[24] D. Burshtein and I. Goldenberg, "Improved linear programming decoding of LDPC codes and bounds on the minimum and fractional distance," *IEEE Trans. Inf. Theory*, vol. 57, no. 11, pp. 7386–7402, Nov. 2011.
[25] K. Yang, J. Feldman, and X. Wang, "Nonlinear programming approaches to decoding low-density parity-check codes," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1603–1613, Aug. 2006.
[26] S. C. Draper, J. S. Yedidia, and Y. Wang, "ML decoding via mixed-integer adaptive linear programming decoding," presented at the Int. Symp. Inf. Theory, Nice, France, Jul. 2007.
[27] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Mach. Learn.*, vol. 3, no. 1, pp. 1–123, 2010.
[28] 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 Trans. Image Process.*, vol. 20, no. 3, pp. 681–695, Mar. 2011.
[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 *Proc. Int. Conf. Mach. Learning*, 2011, pp. 169–176.
[30] P. O. Vontobel and R. Koetter, "Towards low-complexity linear-programming decoding," presented at the Int. Symp. Turbo Codes Related Topics, Munich, Germany, Apr. 2006.
[31] D. Burshtein, "Linear complexity approximate LP decoding of LDPC codes: Generalizations and improvements," presented at the Int. Symp. Turbo Codes Related Topics, Lausanne, Switzerland, Sep. 2008.
[32] P. Vontobel, "Interior-point algorithms for linear-programming decoding," presented at the UCSD Workshop Inf. Theory Apps., San Diego, CA, USA, Jan. 2008.
[33] T. Wadayama, "Interior point decoding for linear vector channels based on convex optimization," in *Proc. Int. Symp. Inf. Theory*, Toronto, CA, USA, Jul. 2008, pp. 1493–1497.
[34] T. Wadayama, "An LP decoding algorithm based on primal path-following interior point method," in *Proc. Int. Symp. Inf. Theory*, Seoul, Korea, Jul. 2009, pp. 389–393.
[35] M.-H. N. Taghavi, A. Shokrollahi, and P. H. Siegel, "Efficient implementation of linear programming decoding," *IEEE Trans. Inf. Theory*, vol. 57, no. 9, pp. 5960–5982, Sep. 2011.
[36] H. Liu, W. Qu, B. Liu, and J. Chen, "On the decomposition method for linear programming decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 58, no. 12, pp. 3448–3458, Dec. 2010.
[37] R. G. Jeroslow, "On defining sets of vertices of the hypercube by linear inequalities," *Discrete Math.*, vol. 11, no. 2, pp. 119–124, 1975.
[38] M. Yannakakis, "Expressing combinatorial optimization problems by linear programs," *J. Comput. Syst. Sci.*, vol. 43, no. 3, pp. 441–466, 1991.
[39] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Belmont, MA, USA: Athena Scientific, 2003.
[40] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer-Verlag, 2006.
[41] H. Wang and A. Banerjee, "Online alternating direction method," presented at the Proc. 29th Int. Conf. Machine Learning, Edinburgh, Scotland, U.K., 2012.
[42] G. D. Forney, "Codes on graphs: Normal realizations," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.
[43] J. S. Yedidia, "The alternating direction method of multipliers as a message-passing algorithms," presented at the Talk Delivered Princeton Workshop Count., Inference Optim., Oct. 2011.
[44] A. Marshall, I. Olkin, and B. C. Arnold, *Inequalities: Theory of Majorization and its Applications*. New York, NY, USA: Springer-Verlag, 2009.
[45] S. Barman, X. Liu, S. C. Draper, and B. H. Recht, "Decomposition methods for large-scale linear-programming decoding," presented at the Allerton Conf. Commun., Control Comput., Monticello, IL, USA, Sep. 2011.
[46] M. B. Allen and E. L. Isaacson, *Numerical Analysis for Applied Science*. New York, NY, USA: Wiley-Interscience, 1998.
[47] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," presented at the Intl. Conf. Space-Time Absoluteness, Ambleside, U.K., 2001.
[48] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[49] D. J. C. MacKay, "Encyclopedia of sparse graph codes," [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html
[50] B. K. Butler and P. H. Siegel, "Error floor approximation for LDPC codes in the AWGN channel," presented at the Allerton Conf. Commun., Contr. Comput., Monticello, IL, USA, Sep. 2011.
[51] B. K. Butler and P. H. Siegel, Arxiv Preprint 1202.2826 "Error floor approximation for LDPC codes in the AWGN channel," 2012.
[52] J. Yedidia, W. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *IEEE Trans. Inf. Theory*, vol. 51, no. 7, pp. 2282–2312, Jul. 2005.
[53] P. O. Vontobel, ArXiv e-Prints "Counting in graph covers: A combinatorial characterization of the Bethe entropy function," Nov. 2010.
[54] 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.*, vol. 57, no. 11, pp. 3258–3268, Nov. 2009.
[55] T. Ruozzi, J. Thaler, and S. Tatikonda, "Graph covers and quadratic minimization," presented at the Allerton Conf. Commun., Control Comput., Monticello, IL, USA, Oct. 2009.
[56] J. S. Yedidia, "A taxonomy suggested by Jonathan Yedidia in personal correspondence," Jan. 2012.
[57] M. E. O'Sullivan, "Algebraic constructions of sparse matrices with large girth," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 718–727, Feb. 2006.
[58] M. P. C. Fossorier, "Quasicyclic, low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.
[59] O. Milenkovic, D. Leyba, and N. Kashyap, "Shortened array codes of large girth," *IEEE Trans. Inf. Theory*, vol. 52, no. 8, pp. 3707–3722, Aug. 2006.
[60] X. Liu, S. Draper, and B. Recht, "Suppressing pseudocodewords by penalizing the objective of LP decoding," presented at the Inf. Theory Workshop, Lausanne, Switzerland, Sep. 2012.

Siddharth Barman is currently doing post-doctorate research in approximation algorithms, combinatorial optimization, and game theory at the California Institute of Technology. Siddharth completed his Ph.D. in Computer Sciences at the University of Wisconsin-Madison in July 2012.

Xishuo Liu obtained his Bachelor's degree from Tsinghua University in 2009. He is currently pursuing a Ph.D. degree in the department of Electrical and Computer Engineering at the University of Wisconsin, Madison, under the supervision of Professor Stark C. Draper.

Stark C. Draper (S'99–M'03) received the M.S. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), and the B.S. and B.A. degrees in electrical engineering and history, respectively, from Stanford University.

He is an Associate Professor of Electrical and Computer Engineering at the University of Toronto, Canada. From 2007–2013 he was an Assistant Professor and an Associate Professor at the University of Wisconsin, Madison. Before moving to the University of Wisconsin he was with the Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA. He has held postdoctoral positions in the Wireless Foundations, University of California, Berkeley, and in the Information Processing Laboratory, University of Toronto. He has worked at Arraycomm, San Jose, CA, the C. S. Draper Laboratory, Cambridge, MA, and Ktaadn, Newton, MA. His research interests include communication and information theory, error-correction coding, statistical signal processing and optimization, security, and application of these disciplines to computer architecture.

Dr. Draper has received the NSF CAREER Award, the 2010 MERL President's Award, the UW ECE Gerald Holdridge Teaching Award, the MIT Carlton E. Tucker Teaching Award, an Intel Graduate Fellowship, Stanford's Frederick E. Terman Engineering Scholastic Award, and a U.S. State Department Fulbright Fellowship.

Benjamin Recht (M'03) is an Assistant Professor at the University of Wisconsin-Madison and a member of the IEEE.