Efficient projection onto the parity polytope and its application to linear programming decoding

Stark Draper
joint work with Siddharth Barman, Xishuo Liu and Ben Recht

Communications & Signal Processing Seminar
University of Michigan
17 October 2013
Setup: consider a length-$d$ single parity-check code

A length-$d$ binary vector $\mathbf{x}$ is a codeword,

$$\mathbf{x} \in C \text{ if } \left[ \begin{array} {ccc} 1 & 1 & \ldots & 1 \end{array} \right] \mathbf{x} = 0$$

or, equivalently, if

$$\mathbf{x} \in P_d$$

where $P_d = \{ \text{all length-$d$ binary vectors of even weight} \}$

In other words: even-weight vertices of the $d$-dimension hypercube
Goal: efficient projection onto \( \text{conv}(\mathbb{P}_d) \), “parity polytope”

The parity polytope \( \mathbb{P}_d = \text{conv}(\mathbb{P}_d) \), the convex hull of \( \mathbb{P}_d \)

- Number of vertices of \( \mathbb{P}_d \) is \( 2^{d-1} \); if \( d = 31 \) about 1 billion
Goal: efficient projection onto $\text{conv}(\mathbb{P}_d)$, “parity polytope”

The parity polytope $\mathbb{P}_d = \text{conv}(\mathbb{P}_d)$, the convex hull of $\mathbb{P}_d$

- Number of vertices of $\mathbb{P}_d$ is $2^{d-1}$; if $d = 31$ about 1 billion
- The algorithm we develop can project any vector $\mathbf{v} \in \mathbb{R}^d$ onto $\mathbb{P}_d$ in log-linear time, $O(d \log d)$, complexity of sort
The parity polytope $\mathbb{P}_d = \text{conv}(\mathbb{P}_d)$, the convex hull of $\mathbb{P}_d$

- Number of vertices of $\mathbb{P}_d$ is $2^{d-1}$; if $d = 31$ about 1 billion
- The algorithm we develop can project any vector $\mathbf{v} \in \mathbb{R}^d$ onto $\mathbb{P}_d$ in log-linear time, $O(d \log d)$, complexity of sort
- We use the projection to develop a new LP decoding technique via the Alternating Directions Method of Multipliers (ADMM)
Agenda

Background and Problem Setup
- LP decoding formulation: a relaxation of ML

Optimization Framework
- The alternating direction method of multipliers (ADMM)

Technical Core
- Characterizing the parity polytope
- Projecting onto the parity polytope

Experimental results
- Various codes & parameter settings
- Penalized decoder
Maximum likelihood (ML) decoding: memoryless channels

- Given codebook $C$ and received sequence $y$
- ML decoding picks a codeword $x \in C$ to:

$$\text{maximize } \Pr(\text{received } y \mid \text{sent } x)$$

$$\Downarrow$$

$$\text{maximize } \prod_i p_{Y \mid X}(y_i \mid x_i) \text{ subject to } x \in C$$

$$\Downarrow$$

$$\text{maximize } \sum_i \log p_{Y \mid X}(y_i \mid x_i) \text{ subject to } x \in C$$
Maximum likelihood (ML) decoding: binary inputs

- Objective for binary input channel:

\[
\sum_i \log p_{Y|X}(y_i | x_i) \\
= \sum_i \left[ \log \frac{p_{Y|X}(y_i | x_i = 1)}{p_{Y|X}(y_i | x_i = 0)} x_i + \log p_{Y|X}(y_i | x_i = 0) \right]
\]

- \(\gamma_i\) is negative log-likelihood ratio of \(i\)th symbol, e.g., if BSC-\(p\):

\[
\gamma_i = \begin{cases} 
\log \frac{p}{1-p} & \text{if } y_i = 1 \\
\log \frac{1-p}{p} & \text{if } y_i = 0 
\end{cases}
\]

- ML decoding: linear objective, integer constraints

\[
\text{minimize } \sum_i \gamma_i x_i \quad \text{s.t. } x \in \mathcal{C}
\]
Specialize to binary linear codes

\[ \mathbf{x} \in C \text{ iff all parity checks have even parity.} \]

Factor graph:

Parity Checks

\[
\begin{align*}
(x_1, x_2, x_3) & \quad (x_1, x_3, x_4) & \quad (x_2, x_5, x_6) & \quad (x_4, x_5, x_6)
\end{align*}
\]

Codeword Bits

\[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6\]

- Let \( d \times n \) matrix \( P_j \) select variables neighboring \( j \)th parity check
- Examples: \( P_1 \mathbf{x} = (x_1 \; x_2 \; x_3) \), \( P_3 \mathbf{x} = (x_2 \; x_5 \; x_6) \)
- Example:

\[
P_3 \mathbf{x} = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_2 \\ x_5 \\ x_6 \end{bmatrix}
\]
For simplicity: consider graphs of check degree $d$

Example: $d = 3$

Parity Checks

\begin{align*}
(x_1, x_2, x_3) & \quad (x_1, x_3, x_4) & \quad (x_2, x_5, x_6) & \quad (x_4, x_5, x_6)
\end{align*}

Codeword Bits

\begin{align*}
x_1 & \quad x_2 & \quad x_3 & \quad x_4 & \quad x_5 & \quad x_6
\end{align*}

- Let $d \times n$ matrix $P_j$ select variables neighboring $j$th parity check
- Examples: $P_1 \mathbf{x} = (x_1 \ x_2 \ x_3)$, $P_3 \mathbf{x} = (x_2 \ x_5 \ x_6)$
- $\mathbb{P}_d = \{\text{all length-}d \text{ binary vectors of even weight}\}$

**Binary linear codes**

$\mathbf{x} \in \mathcal{C}$ if and only if $P_j \mathbf{x} \in \mathbb{P}_d$ for all $j$. 
Relax $\mathcal{P}_d$ to $\mathcal{PP}_d$ to get a Linear Program (LP)

**ML Decoding:** an integer program with a linear objective

$$\text{minimize} \quad \sum_i \gamma_i x_i$$

subject to $P_j x \in \mathcal{P}_d \quad \forall j$

(and $x \in \{0, 1\}^n$)

**LP Decoding:** relax $\mathcal{P}_d$ to $\mathcal{PP}_d = \text{conv}(\mathcal{P}_d)$ for all $j$

$$\text{minimize} \quad \sum_i \gamma_i x_i$$

subject to $P_j x \in \mathcal{PP}_d \quad \forall j$

and $x \in [0, 1]^n$

Relaxation due to Feldman, Wainwright, Karger 2005
Why care about LP decoding?

**LP decoding vs. Belief Propagation (BP) decoding:**

- **BP** empirically successful, inherently distributed, takes full advantage of spare code structure
  - **but**, no convergence guarantees & BP suffers from error-floor

- **LP** well understood theoretically, has convergence guarantees, not observed to suffer from error-floor, ML certificate property, able to tighten relaxation to approach ML performance
  - **but**, generic LP solvers don’t efficiently exploit code sparsity
Why care about projecting onto $\mathbb{PP}_d$?

**Projecting onto $\mathbb{PP}_d$:** crucial step in solving the LP using the *Alternating Direction Method of Multipliers* (ADMM)

- a classic algorithm (mid-70s), efficient, scalable, distributed, convergence guarantees, numerically robust
- decomposes global problem into local subproblems, recombine iteratively (simple scheduling) to find global solution
- simple form today as objective and constraints all linear

Prior work on low-complexity LP decoding:

- earliest low-complexity LP decoding results (Vontobel & Koetter ’06, ’08) coordinate ascent on “softened” dual
- computational complexity linear in blocklength given good choice of scheduling (Burshtein ’08, ’09)
Background and Problem Setup

- LP decoding formulation: a relaxation of ML

Optimization Framework

- The alternating direction method of multipliers (ADMM)

Technical Core

- Characterizing the parity polytope
- Projecting onto the parity polytope

Experimental results

- Various codes & parameter settings
- Penalized decoder
Fitting LP Decoding into ADMM template

LP Decoding:

\[
\text{minimize } \sum_{i} \gamma_i x_i \\
\text{subject to } P_j x \in \mathbb{P}_d \quad \forall j \\
x \in [0, 1]^n
\]

To formulate as an ADMM associate “replicas” \( z_j \)'s with each edge:

\[
\text{minimize } \sum_{i} \gamma_i x_i \\
\text{subject to } z_j = P_j x \quad \forall j \\
z_j \in \mathbb{P}_d \quad \forall j \\
x \in [0, 1]^n
\]

- Replicas allow us to decompose into small subproblems
Lagrangian formulation

minimize \( \sum_i \gamma_i x_i \) subject to \( z_j = P_j x \) \( \forall j \)
\( z_j \in \mathbb{P}_d \) \( \forall j \)
\( x \in [0, 1]^n \)

Start with regular Lagrangian with multipliers \( \lambda = \{\lambda_1, \lambda_2, \ldots\} \)

\( \gamma^T x + \sum_j \lambda_j^T (P_j x - z_j), \)
minimize \( \sum_i \gamma_i x_i \) subject to \( z_j = P_j x \ \forall j \) 
\( z_j \in \mathbb{PP}_d \ \forall j \) 
\( x \in [0, 1]^n \)

Start with regular Lagrangian with multipliers \( \lambda = \{\lambda_1, \lambda_2, \ldots\} \)

\[ \gamma^T x + \sum_j \lambda_j^T (P_j x - z_j), \]

ADMM works with an augmented Lagrangian:

\[ L_\mu(x, z, \lambda) := \gamma^T x + \sum_j \lambda_j^T (P_j x - z_j) + \frac{\mu}{2} \sum_j \|P_j x - z_j\|_2^2 \]

Effect is to smooth the dual problem, accelerating convergence
Alternating Direction Method of Multipliers

Round-robin update of $\mathbf{x}$ then $\mathbf{z}$ then $\lambda$ until converge:

$$L_\mu(\mathbf{x}, \mathbf{z}, \lambda) := \gamma^T \mathbf{x} + \sum_j \lambda_j^T (P_j \mathbf{x} - \mathbf{z}_j) + \frac{\mu}{2} \sum_j \|P_j \mathbf{x} - \mathbf{z}_j\|_2^2$$

ADMM Update Steps:

$$\mathbf{x}^{k+1} := \arg\min_{\mathbf{x} \in \mathcal{X}} L_\mu(\mathbf{x}, \mathbf{z}^k, \lambda^k)$$

$$\mathbf{z}^{k+1} := \arg\min_{\mathbf{z} \in \mathcal{Z}} L_\mu(\mathbf{x}^{k+1}, \mathbf{z}, \lambda^k)$$

$$\lambda_j^{k+1} := \lambda_j^k + \mu \left( P_j \mathbf{x}^{k+1} - \mathbf{z}_j^{k+1} \right)$$

where

$$\mathcal{X} = [0, 1]^n$$

$$\mathcal{Z} = \mathbb{P}P_d \times \ldots \times \mathbb{P}P_d$$

number of checks

- Updates: msg-passing on a “Forney-style” factor graph
ADMM $x$-Update: turns out to be (almost) averaging

With $z$ and $\lambda$ fixed the $x$-updates are:

$$
\text{minimize } L_\mu(x, z^k, \lambda^k) \text{ subject to } x \in [0, 1]^n \text{ where }
L_\mu(x, z, \lambda) := \gamma^T x + \sum_j \lambda_j^T (P_j x - z_j) + \frac{\mu}{2} \sum_j \|P_j x - z_j\|_2^2
$$
With $z$ and $\lambda$ fixed the $x$-updates are:

\[
\text{minimize } L_\mu(x, z^k, \lambda^k) \quad \text{subject to } x \in [0, 1]^n \text{ where }
\]

\[
L_\mu(x, z, \lambda) := \gamma^T x + \sum_j \lambda_j^T (P_j x - z_j) + \frac{\mu}{2} \sum_j \|P_j x - z_j\|^2_2
\]

Partial derivatives of a quadratic form (and apply box constraints)

\[
\frac{\partial}{\partial x_i} L_\mu(x, z^k, \lambda^k) = 0
\]
ADMM x-Update: turns out to be (almost) averaging

With $z$ and $\lambda$ fixed the $x$-updates are:

$$\text{minimize } L_\mu(x, z^k, \lambda^k) \text{ subject to } x \in [0, 1]^n \text{ where}$$

$$L_\mu(x, z, \lambda) := \gamma^T x + \sum_j \lambda_j^T (P_j x - z_j) + \frac{\mu}{2} \sum_j \|P_j x - z_j\|^2_2$$

Partial derivatives of a quadratic form (and apply box constraints)

$$\frac{\partial}{\partial x_i} L_\mu(x, z^k, \lambda^k) = 0$$

Get component-wise (averaging) updates:

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

$\mathcal{N}_\mathcal{V}(i)$ : set of parity checks neighboring variable $i$.  
$z_j^{(i)}$ : component of the $j$th replica associated with $x_i$.  


Recall:

\[ L_\mu(x, z, \lambda) := \gamma^T x + \sum_j \lambda_j^T (P_j x - z_j) + \frac{\mu}{2} \sum_j \|P_j x - z_j\|_2^2 \]

**z-update**: with \( x \) and \( \lambda \) fixed we want to solve

\[
\text{minimize} \quad \sum_j \lambda_j^T (P_j x - z_j) + \frac{\mu}{2} \sum_j \|P_j x - z_j\|_2^2 \\
\text{subject to} \quad z_j \in \mathbb{P}_d \quad \forall j
\]

The minimization is separable in \( j \): for each \( j \) we need to solve

\[
\text{minimize} \quad \lambda_j^T (P_j x - z_j) + \frac{\mu}{2} \|P_j x - z_j\|_2^2 \\
\text{subject to} \quad z_j \in \mathbb{P}_d
\]
ADMM $z_j$-Update: project onto parity polytope

$z_j$-update:

minimize $\lambda_j^T(P_j x - z_j) + \frac{\mu}{2} \|P_j x - z_j\|_2^2$

subject to $z_j \in \mathbb{PP}_d$

Setting $v = P_j x + \lambda_j / \mu$ (completing the square) the problem is equivalent to:

minimize $\|v - \tilde{z}\|_2^2$

subject to $\tilde{z} \in \mathbb{PP}_d$

The primary challenge in ADMM

The $z$-update requires projecting onto the parity polytope.
Background and Problem Setup
- LP decoding formulation: a relaxation of ML

Optimization Framework
- The alternating direction method of multipliers (ADMM)

Technical Core
- Characterizing the parity polytope
- Projecting onto the parity polytope

Experimental results
- Various codes & parameter settings
- Penalized decoder
Prior characterizations of parity polytope

- Jeroslow (1975)
- Yannakakis (1991) has a quadratic $d^2$ characterization
- Feldman et al. (2005) use Yannakakis
- “Standard Polytope” in Feldman uses $2^{d-1}$ linear constraints per parity-check, many not active as exploited in “Adaptive LP Decoding” Taghavi and Siegel (2008)
Most points in $\mathbb{PP}_d$ have multiple representations

By definition:

- $y \in \mathbb{PP}_d$ iff $y = \sum_i \alpha_i e_i$
- $\sum_i \alpha_i = 1$, $\alpha_i \geq 0$
- $e_i$ are even-hamming-weight binary vectors of dimension $d$
- Most $y \in \mathbb{PP}_d$ have multiple representations

Example A ($d = 6$):

\[
\begin{pmatrix}
1 \\
1 \\
1/2 \\
1/2 \\
1/4 \\
1/4
\end{pmatrix}
= \frac{1}{2}
\begin{pmatrix}
1 \\
1 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}
+ \frac{1}{4}
\begin{pmatrix}
1 \\
1 \\
1 \\
0 \\
0 \\
0
\end{pmatrix}
+ \frac{1}{4}
\begin{pmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1
\end{pmatrix}
\]
Most points in \( PP_d \) have multiple representations

By definition:

- \( y \in PP_d \) iff \( y = \sum_i \alpha_i e_i \)
- \( \sum_i \alpha_i = 1, \alpha_i \geq 0 \)
- \( e_i \) are even-hamming-weight binary vectors of dimension \( d \)
- Most \( y \in PP_d \) have multiple representations

Example B \((d = 6)\):

\[
\begin{pmatrix}
1 \\
1 \\
1/2 \\
1/2 \\
1/4 \\
1/4
\end{pmatrix}
= \frac{1}{4}
\begin{pmatrix}
1 \\
1 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}
+ \frac{1}{2}
\begin{pmatrix}
1 \\
1 \\
1 \\
1 \\
0 \\
0
\end{pmatrix}
+ \frac{1}{4}
\begin{pmatrix}
1 \\
1 \\
0 \\
0 \\
1 \\
1
\end{pmatrix}
\]
There always exists a “two-slice” representation

Two-Slice Lemma:
For any $y \in \mathbb{P}P_d$ there exists a representation $y = \sum_i \alpha_i e_i$ where
- $\sum_i \alpha_i = 1$, $\alpha_i \geq 0$
- $e_i$ are of only two weights: $r$ or $r + 2$ for all $i$
- $r$ is the even integer $r = \lfloor \|y\|_1 \rfloor_{\text{even}}$

Example B is one such representation with $d = 6$ and $r = 2$:

$$
\begin{pmatrix}
1 \\
1 \\
1/2 \\
1/2 \\
1/4 \\
1/4
\end{pmatrix}
= \frac{1}{4}
\begin{pmatrix}
1 \\
1 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}
+ \frac{1}{2}
\begin{pmatrix}
1 \\
1 \\
1 \\
0 \\
0 \\
0
\end{pmatrix}
+ \frac{1}{4}
\begin{pmatrix}
1 \\
0 \\
0 \\
1 \\
0 \\
1
\end{pmatrix}
$$
Visualizing properties of $\mathbb{P}P_d$: always between two slices

Example: $d = 5$

Let $\mathbb{P}P'_d = \text{conv}\{e_i \mid \|e_i\|_1 = r\}$, a “permutohedron”
\[ \Rightarrow \] easy to characterize using majorization

- **Two-slice restated**: Any $y \in \mathbb{P}P_d$ is sandwiched between two permutohedrons $\mathbb{P}P'_d$ and $\mathbb{P}P_{d+r}^2$ where $r = \left\lfloor \|y\|_1 \right\rfloor_{\text{even}}$
Majorization: definition & application to $\mathbb{PP}_d^r$

**Definition:** Let $\mathbf{u}$ and $\mathbf{w}$ be $d$-vectors sorted in decreasing order. The vector $\mathbf{w}$ is said to majorize $\mathbf{u}$ if

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

Specialize to $\mathbb{PP}_d^r$ where $\mathbf{w} = [1 \ldots 1 \ 0 \ 0 \ldots 0]$

\[
\sum_{k=1}^{d} u_k = r \\
\sum_{k=1}^{q} u_k \leq \min(q, r) \quad \forall \quad q, 1 \leq q < d
\]
Theorem: \( u \) is in the convex hull of all permutations of \( w \) (the permutohedron defined by \( w \)) if and only if \( w \) majorizes \( u \).
**Theorem:** \( u \) is in the convex hull of all permutations of \( w \) (the permutohedron defined by \( w \)) if and only if \( w \) majorizes \( u \).

\[
    u = \sum_{i} \beta_i \Sigma_i w
\]

where \( \Sigma_i \) are permutation matrices and \( \beta_i \) are weightings.
Theorem: \( \mathbf{u} \) is in the convex hull of all permutations of \( \mathbf{w} \) (the permutohedron defined by \( \mathbf{w} \)) if and only if \( \mathbf{w} \) majorizes \( \mathbf{u} \).

\[
\mathbf{u} = \sum_{i} \beta_{i} \Sigma_{i} \mathbf{w}
\]

where \( \Sigma_{i} \) are permutation matrices and \( \beta_{i} \) are weightings.

Proving two-slice lemma:
- Use above to characterize each \( \mathcal{PP}_{d}^{r} \), \( r \) even, \( r \leq d \).
- Express \( \mathbf{y} \) as a weighted combination of points in \( \mathcal{PP}_{d}^{r} \), \( 1 \leq r \leq d \).
- Show you can set all weightings to zeros except those on \( r = \lceil \| \mathbf{y} \|_{1} \rceil_{\text{even}} \) and \( r = \lceil \| \mathbf{y} \|_{1} \rceil_{\text{even}} + 2 \).
- Note that finding \( r \) is trivial.

Next: use two-slice lemma to develop projection operation.
Projecting onto the parity polytope

Desired projection:

\[
\begin{align*}
\min & \quad \|v - y\|_2^2 \\
n & \quad y \in \mathbb{PP}_d
\end{align*}
\]
Projecting onto the parity polytope

Desired projection:

\[
\min \|v - y\|^2_2 \\
\text{s.t. } y \in \mathbb{PP}_d
\]

Use two-slice lemma to reformulate as:

\[
\min \|v - \alpha s - (1 - \alpha)t\|^2_2 \\
\text{s.t. } 0 \leq \alpha \leq 1, \ s \in \mathbb{PP}_d^r, \ t \in \mathbb{PP}_d^{r+2}
\]
Projecting onto the parity polytope

Desired projection:

\[
\min \|v - y\|_2^2
\]  
\[
\text{s.t. } y \in \mathbb{PP}_d
\]

Use two-slice lemma to reformulate as:

\[
\min \|v - \alpha s - (1 - \alpha)t\|_2^2
\]  
\[
\text{s.t. } 0 \leq \alpha \leq 1, \ s \in \mathbb{PP}_d^r, \ t \in \mathbb{PP}_d^{r+2}
\]

We also show (where \(\Pi(\cdot)\) is shorthand for projection):

\[
\left\lfloor \frac{\|\Pi_{[0,1]^d}(v)\|_1}{r} \right\rfloor_{\text{even}} \leq \|\Pi_{\mathbb{PP}_d}(v)\|_1 \leq \left\lfloor \frac{\|\Pi_{[0,1]^d}(v)\|_1}{r+2} \right\rfloor_{\text{even}} + 2
\]

in other words, it is trivial to identify the two slices
Use majorization to simplify problem further

Assume w.l.o.g that \( \mathbf{v} \) is sorted and let

\[
\mathbf{z} = \Pi_{PP_d}(\mathbf{v}) = \arg\min \|\mathbf{v} - \alpha \mathbf{s} - (1 - \alpha) \mathbf{t}\|_2^2
\]

\[
\text{s.t. } 0 \leq \alpha \leq 1, \quad \mathbf{s} \in PP_d, \quad \mathbf{t} \in PP_d^{r+2}
\]

Constraint set can be restated as

\( (i) \) \( \quad 0 \leq \alpha \leq 1 \)

\( (ii) \) \( \quad \sum_{k=1}^{d} z_k = \alpha r + (1 - \alpha)(r + 2) \)

\( (iii) \) \( \quad \sum_{k=1}^{q} z_k \leq \alpha \min(q, r) + (1 - \alpha) \min(q, r + 2) \quad \forall \quad q, \quad 1 \leq q < d \)

\( (iv) \) \( \quad z_1 \geq z_2 \geq \ldots \geq z_d \)
Combine knowledge of $r$ with first two constraints

From (ii) we have

$$
\sum_{k=1}^{d} z_k = \alpha r + (1 - \alpha)(r + 2)
$$

(*)

Now we apply the bound from (i) on $\alpha$, $0 \leq \alpha \leq 1$ to get

$$
r \leq \sum_{k=1}^{d} z_k \leq r + 2
$$
Deal with third constraint

Consider the partial sums of the sorted vectors

\[
\sum_{k=1}^{q} z_k \leq \alpha \min(q, r) + (1 - \alpha) \min(q, r + 2) \quad \forall \quad q, 1 \leq q < d
\]

- For \( q \leq r \) ineq. satisfied by box constraints: \( 0 \leq z_k \leq 1 \quad \forall k \)
- For \( q \geq r + 2 \) inequalities also satisfied since

\[
\sum_{k=1}^{q} z_k \leq \sum_{k=1}^{d} z_k = \alpha r + (1 - \alpha)(r + 2)
\]

Hence only need to deal with \( q = r + 1 \), which specializes as

\[
\sum_{k=1}^{r+1} z_k \leq \alpha r + (1 - \alpha)(r + 1) = r + (1 - \alpha)
\]
Third constraint (continued...)

Solve (*) for $\alpha$ to find

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

Finally, substitute into (**) to get

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

$$= r - \frac{r - \sum_{k=1}^{d} z_k}{2}$$

Which becomes

$$\sum_{k=1}^{r+1} z_k - \sum_{k=r+2}^{d} z_k \leq r$$
Reformulated projection as a quadratic program (QP)

\[
\begin{align*}
\text{min} & \quad \|v - \alpha s - (1 - \alpha)t\|_2^2 \\
\text{s.t.} & \quad 0 \leq \alpha \leq 1 \\
& \quad s \in \mathbb{PP}_d^r, \\
& \quad t \in \mathbb{PP}_{d+r+2} \\
\end{align*}
\]

\[
\begin{align*}
\text{min} & \quad \|v - z\|_2^2 \\
\text{s.t.} & \quad 1 \geq z_k \geq 0 \ \forall \ k \\
& \quad z_1 \geq z_2 \geq \ldots \geq z_d \\
& \quad r + 2 \geq \sum_k z_k \geq r \\
& \quad r \geq \sum_{k=1}^{r+1} z_k - \sum_{k=r+2}^d z_k
\end{align*}
\]
Reformulated projection as a quadratic program (QP)

\[
\begin{align*}
\min \| &v - \alpha s - (1 - \alpha)t\|^2_2 \\
\text{s.t.} \quad &0 \leq \alpha \leq 1 \\
&s \in \mathbb{PP}_d^r, \\
t \in \mathbb{PP}_d^{r+2}
\end{align*}
\]

\[
\begin{align*}
\min \| &v - z\|^2_2 \\
\text{s.t.} \quad &1 \geq z_k \geq 0 \quad \forall \ k \\
&z_1 \geq z_2 \geq \ldots \geq z_d \\
&r + 2 \geq \sum_{k} z_k \geq r \\
&r \geq \sum_{k=1}^{r+1} z_k - \sum_{k=r+2}^{d} z_k
\end{align*}
\]

- for the QP the KKT conditions are necessary and sufficient
- we develop a linear-time water-filling type algorithm that determines a solution satisfying the KKT conditions

\[
z^* = \prod_{[0,1]^d} \left( v - \beta \left[ \begin{array}{c} 1 \ldots 1 \\ r+1 \end{array} \right] - \left[ \begin{array}{c} 1 \ldots -1 \\ d-r-1 \end{array} \right] \right) \quad \text{some} \quad \beta_{\text{opt}} \in [0, \beta_{\text{max}}]
\]
Agenda

Background and Problem Setup
- LP decoding formulation: a relaxation of ML

Optimization Framework
- The alternating direction method of multipliers (ADMM)

Technical Core
- Characterizing the parity polytope
- Projecting onto the parity polytope

Experimental results
- Various codes & parameter settings
- Penalized decoder
Performance results: two LDPC codes over AWGN

- length-2640, rate-0.5
- (3, 6)-regular LDPC
- *non-saturating* BP per Butler & Siegel (Allerton '11)

- length-1057, rate-0.77
- (3, 13)-regular LDPC
- observable error floor
Performance results: random LDPC ensemble over BSC

- results averaged over ensemble of 100 codes
- each a randomly generated length-1002 (3, 6)-regular LDPC
- all codes had girth at least 4
Random ensemble: iteration count & execution time

- iteration count
- ADMM & BP for:
  (i) errors, (ii) avg, (iii) correct

- execution time
- ADMM & BP for
  (i) errors, (ii) avg, (iii) correct
Understanding LP decoding failures

LP decoding fails to a “pseudocodeword”, a non-integer vertex of the fundamental polytope introduced when we relaxed each of the various integer constraints $P_d$ to $PP_d$ in

$$\min \gamma^T x \text{ s.t. } P_j x \in PP_d \forall j, \ x \in [0, 1]^n$$
\( \ell_2 \)-penalized ADMM

In order to eliminate pseudocodewords, introduce an \( \ell_2 \)-penalty to push the solution towards an integral solution, now solve:

\[
\min \gamma^T x - c \| x - 0.5 \|_2 \quad \text{s.t.} \quad P_j x \in \mathbb{PP}_d \quad \forall j, \quad x \in [0,1]^n
\]
\(\ell_2\)-penalized ADMM

In order to eliminate pseudocodewords, introduce an \(\ell_2\)-penalty to push the solution towards an integral solution, now solve:

\[
\min \gamma^T x - c \|x - 0.5\|_2 \quad \text{s.t.} \quad P_j x \in \mathbb{PP}_d \quad \forall j, \quad x \in [0, 1]^n
\]

[2640,1320] “Margulis” LDPC

[13298, 3296] rate-0.25 LDPC
Recap & wrap-up

Recap:

- LP decoding via ADMM
- main hurdle: efficient projection onto the parity polytope, complexity of sort
- simple scheduling and complexity linear in the block-length
- roughly same execution time as BP
- further improvements via $\ell_2$-penalty (alternately $\ell_1$-penalty)
- **Try it yourself!** Documented code available at https://sites.google.com/site/xishuoliu/codes

Things to do:

- error floor analysis (LP & penalized)
- effects of finite precision
- how to implement in hardware
- understand BP/LP low-SNR gap (without penalty)
- other codes: non-binary codes, permutation-based codes
2014 IEEE North American School on Information Theory

To be held at the Fields Institute at the University of Toronto
18-21 June 2014