Decoding LDPC Codes by Using Negative Proximal Regularization

Yiming Chen, Rui Wang, Jinglong Zhu, and Zaiwen Wen

***Abstract*— The low-density parity-check (LDPC) decoding problem can be expressed as an integer linear programming (ILP) problem. One efficient method to solve the ILP problem is to relax the integer constraints and add penalty terms to the objective function, and the revised problem can be solved via the alternating direction method of multipliers (ADMM) algorithm. These penalty terms can punish the non-integral solutions and improve the decoding performance of the decoder. However, ADMM decoders are easily trapped in a local solution, which limits the frame error rate (FER) performance of the decoders at low signal-to-noise ratios (SNR). In this paper, we propose a restartable ADMM-based decoder using a negative proximal regularization. The negative proximal term will be updated whenever the decoder finds a new local solution. Therefore, the decoder can be restarted several times and the candidate solution which satisfies the parity-check equations and has the lowest objective function value can be selected as the decoder's output. Some properties, together with several choices of penalty terms are discussed. We also investigate the convergence of our proposed decoder, and prove that the possibility of decoding errors is independent of the codeword that is transmitted. Simulation results show that our proposed decoder outperforms other ADMM-based decoders in most cases, while the decoding complexity maintains the same.**

***Index Terms*— LDPC, ADMM, negative proximal regularization, restartable decoder.**

I. Introduction

LOW-DENSITY parity-check (LDPC) codes were first introduced by [1] in 1962, and they were rediscovered by [2] and [3] in 1990s. The decoders for LDPC codes were most commonly based on belief propagation (BP), including the sum-product decoder and other variants [4], [5]. When using BP decoding, LDPC codes can perform near the Shannon limit. However, in the high signal-to-noise ratio (SNR) region, BP decoding often suffers the "error floor" phenomenon, which refers that the error-rate cannot drop rapidly as the SNR increases.

Manuscript received 2 October 2022; revised 14 March 2023 and 24 April 2023; accepted 28 April 2023. Date of publication 9 May 2023; date of current version 17 July 2023. The work of Zaiwen Wen was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 11831002. The associate editor coordinating the review of this article and approving it for publication was A. E. Pusane. *(Corresponding author: Rui Wang.)*

Yiming Chen, Rui Wang, and Zaiwen Wen are with the Beijing International Center for Mathematical Research, and Changsha Institute for Computing and Digital Economy, Peking University, Beijing 100871, China (e-mail: abcdcym@stu.pku.edu.cn; ruiwang@bicmr.pku.edu.cn; wenzw@pku.edu.cn).

Jinglong Zhu is with Huawei Beijing Research Center, Beijing 100095, China (e-mail: zhujinglong@huawei.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCOMM.2023.3274150.

Digital Object Identifier 10.1109/TCOMM.2023.3274150

The decoding problem can be equivalently expressed as an integer linear programming (ILP) problem, and then solved by linear programming (LP) relaxation. LP decoding for LDPC codes was first introduced by [6] and another method to transform the decoding problem to an LP problem was later proposed by [7]. Compared with BP decoding, LP decoding has several outstanding attributes. Firstly, LP decoding has a better decoding performance at a high SNR level. Secondly, LP decoding has the "maximum-likelihood (ML) certificate" property, which refers that the integral solution to the LP decoding problem is also the ML solution. One drawback of LP decoding is the high computation consumption. In [8], [9] and [10], several algorithms were proposed to reduce the complexity of the LP decoders. In [10], the alternating direction method of multipliers (ADMM) technique was introduced to the LP decoders. The ADMM decoder has closed-form solutions in each iteration, which significantly improves the efficiency of LP decoding.

In the past decade, there have been two directions to improve ADMM decoding. One direction is to accelerate the decoding process. In the ADMM decoding process, an Euclidean projection onto the parity-check polytope is to be conducted in each iteration, which dominates the complexity of ADMM decoding. Therefore, many works [11], [12], [13], [14], [15], [16] focus on reducing the complexity of the projection process. In [15], an iterative projection algorithm was proposed, which had a linear complexity in the worst case. In [16], a line segment projection algorithm was introduced, which reduced the projection time evidently without iterative operations. There are also some works improving the efficiency of ADMM decoding from other aspects. In [17], the authors proposed a check-polytope-free ADMM framework for the LP model in [7], where a linear complexity in terms of the length of codewords can be guaranteed.

The other direction is to improve the decoding performance. Since the LP model relaxes the binary constraints to continuous ones, the optimal solution to the LP decoding problem may be fractional. The ADMM decoder fails whenever a non-integral solution, which is called pseudo-codeword, is output. In [18], the authors added a penalty term to the objective function of the LP model. The penalty term is a symmetric function that takes smaller values at integral points than that at fractional ones. Thus the decoder is more likely to output an integral solution. The authors also investigated several types of penalty functions and compared their decoding performance. Reference [19] used a segmented function as

the penalty term, which gained improvement in decoding performance. In [20], a new penalty term was proposed, which penalized the check nodes instead of the variable nodes. Additionally, some authors improved the decoding performance by introducing new constraints that play the same role as the penalty term. Reference [21] introduced the ℓ_p-box constraints, while [22] added an auxiliary variable and introduced constraints that force the variable to be 0 or 1.

In this paper, we focus on improving the frame-error-rate (FER) performance of ADMM decoding at low SNRs. Since there is no guarantee that the ADMM decoding algorithm can converge to the ML solution, the ADMM iterations may be trapped in a local solution or a pseudo-codeword. Therefore, we propose a restartable ADMM framework, which is obtained by adding a negative proximal term to the objective function of the LP model. Whenever the ADMM algorithm converges to a stationary point, the negative proximal term will be adjusted such that the ADMM algorithm can be restarted. The negative proximal term suppresses the stationary points found during the decoding process, pushing the algorithm to converge to a new local minimum. There are many choices of negative proximal terms, and we investigate the ℓ_1-norm and ℓ_2-norm-square penalty functions in this paper.

There are two major differences between our proposed decoder and other existing ADMM penalized decoders. First, the penalty terms are different. We choose a symmetric function that has a peak at a given point $\hat{\mathbf{x}}$ as the penalty function. The point $\hat{\mathbf{x}}$ is initialized as a vector whose entries are all 0.5. Thus our objective function is the same as that of the ADMM penalized decoder [18] at the beginning of the decoding process. However, after the ADMM algorithm finds a stationary point, $\hat{\mathbf{x}}$ will be updated and our model will have a different form from other decoders. Second, the decoding processes are distinct. For other ADMM-based decoders, the ADMM algorithm will be executed once, and only one solution will be output by the decoder, even if the solution is not a codeword. For our proposed decoder, the ADMM algorithm will be conducted by several times, and several candidate solutions will be found during the decoding process. The optimal solution with the smallest objective function value can be selected as the output of our decoder.

The main contributions in this paper are listed as follows:

- We propose a restartable ADMM decoding framework by adding a negative proximal term to the LP decoding model. We investigate several types of penalty functions as the negative proximal term and design penalty functions with weighted forms for irregular LDPC codes.
- We conduct theoretical analysis including the convergence property of the ADMM algorithm and the all-zero assumptions of our proposed decoder.

The rest of the paper is organized as follows. In Section II, we review the LP decoding model and the ADMM penalized decoder. Then we introduce our restartable decoding model. In Section III, we present the ADMM algorithm to solve our decoding model and introduce the restartable decoding framework. Section IV includes the analysis of convergence and other properties of our decoder. Simulation results, which show the performance improvement of our proposed decoder, are presented in Section V. Section VI makes a conclusion of this paper.

II. Preliminaries

Throughout this paper, we consider LDPC codes $\mathcal{C}$ of length n, each specified by an $m \times n$ parity check matrix $\mathbf{H}$. Let $\mathcal{I} = \{1, 2, \ldots, m\}$ and $\mathcal{J} = \{1, 2, \ldots, n\}$ denote the sets of check nodes and variable nodes of $\mathcal{C}$, respectively. Suppose a codeword $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathcal{C}$ is transmitted over a noisy memoryless binary-input output-symmetric channel, resulting in a corrupted signal $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$. The corresponding log-likelihood ratios are defined as

$$\gamma_j = \log\left(\frac{\Pr(y_j|x_j = 0)}{\Pr(y_j|x_j = 1)}\right), \quad j \in \mathcal{J}. \tag{1}$$

A. ML Decoding Problem

The ML decoding problem can be formulated as the following integer programming problem [6]:

$$\min \ \boldsymbol{\gamma}^T\mathbf{x}, \tag{2a}$$
$$\text{s.t. } [\mathbf{H}\mathbf{x}]_2 = 0, \tag{2b}$$
$$\mathbf{x} \in \{0, 1\}^n, \tag{2c}$$

where $\boldsymbol{\gamma}$ is an n-dimensional vector composed of the above log-likelihood ratios and $[.]_2$ denotes the modular-2 operator.

The difficulty of solving the ML decoding problem (2) comes from the parity check constraints (2b) and the discrete constraints (2c). In [6], J. Feldman et al. relaxed these constraints and proposed an LP form of the decoding problem.

B. LP Relaxation

Denote the set $\mathcal{C}^i = \{\mathbf{x} \in \{0, 1\}^n : [\mathbf{h}_i\mathbf{x}]_2 = 0\}$, where $\mathbf{h}_i$ is the i-th row of $\mathbf{H}$. Then the integer programming problem (2) can be relaxed to the following form

$$\min \ \boldsymbol{\gamma}^T\mathbf{x}, \tag{3a}$$
$$\text{s.t. } \mathbf{x} \in \cap_{i\in\mathcal{I}} \operatorname{conv}(\mathcal{C}^i), \tag{3b}$$

where $\operatorname{conv}(\mathcal{C}^i)$ denotes the convex hull of $\mathcal{C}^i$. We now introduce the notation $\mathbf{P}_i$ as the $d_i \times N$ binary selection matrix which selects the sub-vector of $\mathbf{x}$ that participates the i-th parity-check equation, where d_i denotes the number of 1s in $\mathbf{h}_i$. An equivalent expression of (3) is

$$\min \ \boldsymbol{\gamma}^T\mathbf{x}, \tag{4a}$$
$$\text{s.t. } \mathbf{P}_i\mathbf{x} \in \mathcal{PP}_{d_i}, \quad \forall i \in \mathcal{I}. \tag{4b}$$

Here $\mathcal{PP}_{d_i}$ is the parity polytope of dimension d_i, which is the convex hull of all binary vectors that have even weights. Solving (4) directly by a general LP optimizer is not efficient. [10] reformulated (4) as the following problem and solved it by the ADMM algorithm:

$$\min \ \boldsymbol{\gamma}^T\mathbf{x}, \tag{5a}$$
$$\text{s.t. } \mathbf{P}_i\mathbf{x} = \mathbf{z}_i, \tag{5b}$$
$$\mathbf{z}_i \in \mathcal{PP}_{d_i}, \quad \forall i \in \mathcal{I}, \tag{5c}$$

where $\mathbf{z}_i$ is a d_i-dimensional vector. The introduction of auxiliary variable $\mathbf{z}_i$ makes the model fit the ADMM algorithm.

C. Penalized Decoding Model

LP decoding fails whenever it outputs a fractional solution which is called a pseudo-codeword. Therefore, [18] tightened the constraints (4b) by adding the penalty term $g(x_j)$ for each variable x_j to the objective function (4a) and proposed an ADMM penalized model. This model is given as follows:

$$\min \ \boldsymbol{\gamma}^T\mathbf{x} + \sum\nolimits_{j\in\mathcal{J}} g(x_j), \tag{6a}$$

$$\text{s.t. } \mathbf{P}_i\mathbf{x} = \mathbf{z}_i, \tag{6b}$$

$$\mathbf{z}_i \in \mathcal{PP}_{d_i}, \quad \forall i \in \mathcal{I}. \tag{6c}$$

The penalty function $g(\mathbf{x})$ is an increasing function on $[0, 0.5]$, and is symmetric about 0.5. This makes the fractional solutions to this model more costly than the integral ones.

D. Restartable Decoding Model

Though the penalty function penalizes the fractional solution of (6) efficiently, there is no guarantee that the ADMM algorithm for (6) can converge to the global minimum. In this paper, we propose a restartable decoding algorithm, which can be restarted whenever the algorithm converges to a local minimum. To achieve this, we introduce a new penalty function $f(\mathbf{x})$ to the model (6). We are partially motivated by [23], in which the authors added a negative proximal term to the objective function to obtain an improvement of the performance. We also discuss some properties of the penalty function. Then several examples of the penalty functions and their variable update rules are presented.

We formally state the restartable ADMM decoding problem as follows:

$$\min \ \boldsymbol{\gamma}^T\mathbf{x} - \rho f(\mathbf{x} - \hat{\mathbf{x}}), \tag{7a}$$

$$\text{s.t. } \mathbf{P}_i\mathbf{x} = \mathbf{z}_i, \tag{7b}$$

$$\mathbf{z}_i \in \mathcal{PP}_{d_i}, \quad \forall i \in \mathcal{I}. \tag{7c}$$

In (7), ρ is the penalty parameter, and $f(\mathbf{x}-\hat{\mathbf{x}})$ is the penalty term. $\hat{\mathbf{x}}$ is set as the average of 0.5 and several stationary points that have been reached during the iterations. Thus the penalty term plays a role in pushing the algorithm to find another integral solution which is far away from all the existing solutions. The penalty function $f = \sum_{j\in\mathcal{J}} f_j(x_j)$, where each f_j should have the following properties:

- f_j is an increasing function on $[0, 1]$.
- f_j is symmetric about 0.
- f_j is non-negative and $f_j(x) > 0$ at some $x \in [0, 1]$.

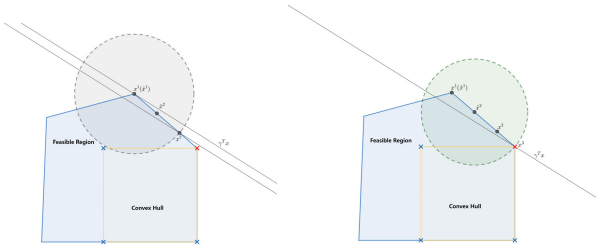
Let ν be any positive constant satisfying

$$|\boldsymbol{\gamma}^T\mathbf{x} - \boldsymbol{\gamma}^T\mathbf{y}| \geq \nu, \quad \forall \mathbf{x}, \mathbf{y} \in \mathcal{C}, \ \boldsymbol{\gamma}^T\mathbf{x} \neq \boldsymbol{\gamma}^T\mathbf{y}, \tag{8}$$

where $\mathcal{C}$ denotes the set of all codewords. Since $\mathcal{C}$ is a finite set, we can always find a proper ν satisfying the condition (8). Then the next theorem shows that the integral solution of (7) is also a solution of problem (2) when the parameter ρ is chosen suitably.

Theorem 1: Suppose that the penalty parameter satisfies $0 < \rho < \frac{\nu}{M}$, *where* $M = \max_{\mathbf{x}\in[-1,1]^n} f(\mathbf{x})$. *Then any solution* $\mathbf{x}^*_\rho$ *of (7) is also optimal for (2). Moreover, for any solution* $\mathbf{x}^*$ *of (2), we have*

$$f(\mathbf{x}^*_\rho - \hat{\mathbf{x}}) \geq f(\mathbf{x}^* - \hat{\mathbf{x}}). \tag{9}$$



(a) The first restart. (b) The second restart.

Fig. 1. A simple illustration of our proposed restartable decoding process, where the feasible solutions and the optimal solution are denoted as the blue crosses and red cross, respectively.

Proof: From the optimality and feasibility of $\mathbf{x}^*_\rho$, we have

$$\boldsymbol{\gamma}^T\mathbf{x}^*_\rho - \rho f(\mathbf{x}^*_\rho - \hat{\mathbf{x}}) \leq \boldsymbol{\gamma}^T\mathbf{x}^* - \rho f(\mathbf{x}^* - \hat{\mathbf{x}}), \tag{10}$$

which implies that

$$\begin{aligned}\boldsymbol{\gamma}^T\mathbf{x}^*_\rho &\leq \boldsymbol{\gamma}^T\mathbf{x}^* + \rho(f(\mathbf{x}^*_\rho - \hat{\mathbf{x}}) - f(\mathbf{x}^* - \hat{\mathbf{x}})) \\ &\leq \boldsymbol{\gamma}^T\mathbf{x}^* + \rho M < \boldsymbol{\gamma}^T\mathbf{x}^* + \nu.\end{aligned}$$

Moreover, the optimality of $\mathbf{x}^*$ yields that $\boldsymbol{\gamma}^T\mathbf{x}^* - \boldsymbol{\gamma}^T\mathbf{x}^*_\rho \leq 0 < \nu$. Thus we have $|\boldsymbol{\gamma}^T\mathbf{x}^*_\rho - \boldsymbol{\gamma}^T\mathbf{x}^*| < \nu$. Together with the assumption (8), it follows that

$$\boldsymbol{\gamma}^T\mathbf{x}^* = \boldsymbol{\gamma}^T\mathbf{x}^*_\rho. \tag{11}$$

From (10) and (11), the inequality (9) can be obtained immediately. ■

Fig. 1 gives an explanation of our proposed restartable decoding process. We apply our model to solve a two-dimensional ILP problem. We use the crosses to denote all the feasible solutions. Additionally, we use the blue polygon and the orange polygon to represent the feasible region of the LP relaxation and the convex hull of the feasible solutions, respectively. The red cross is the optimal solution of this problem. Our decoding process starts at the optimal solution of the LP relaxation problem, which is denoted as $\mathbf{x}^1$. As shown in Fig. 1(a), we set $\hat{\mathbf{x}}^1$ as $\mathbf{x}^1$ at the first restart, and the gray circle centered on $\hat{\mathbf{x}}^1$ will be cut due to the negative proximal term. Therefore, a new solution $\mathbf{x}^2$ will be found. In Fig. 1(b), we set $\hat{\mathbf{x}}^2$ as the average of $\mathbf{x}^1$ and $\mathbf{x}^2$. The green circular region centered on $\hat{\mathbf{x}}^2$ will be cut after the second restart, and the decoder will find the third solution $\mathbf{x}^3$, which is exactly the optimal solution of the ILP problem.

III. Restartable ADMM Decoding

In this section, we present the restartable ADMM decoding framework. We discuss several choices of the negative proximal term and design a weighted penalty function for irregular LDPC codes. We also propose an algorithm to produce a feasible codeword from a given binary vector, which can be implemented as a post-treatment of our decoder to improve the decoding performance.

A. ADMM Decoding Algorithm

We next present the ADMM algorithm for solving problem (7). The augmented Lagrangian function is

$$L(\mathbf{x},\mathbf{z},\boldsymbol{\lambda}) = \boldsymbol{\gamma}^T\mathbf{x} - \rho f(\mathbf{x}-\hat{\mathbf{x}}) + \sum\nolimits_i \boldsymbol{\lambda}_i^T(\mathbf{P}_i\mathbf{x}-\mathbf{z}_i) + \frac{\mu}{2}\sum\nolimits_i \|\mathbf{P}_i\mathbf{x}-\mathbf{z}_i\|_2^2. \quad (12)$$

The iterative scheme of ADMM for this problem is given by

$$\mathbf{z}^{k+1} = \arg\min_{\mathbf{z}_i\in\mathcal{PP}_{d_i}} L(\mathbf{x}^k,\mathbf{z},\boldsymbol{\lambda}^k), \quad (13a)$$

$$\mathbf{x}^{k+1} = \arg\min_{\mathbf{x}\in[0,1]^n} L(\mathbf{x},\mathbf{z}^{k+1},\boldsymbol{\lambda}^k), \quad (13b)$$

$$\boldsymbol{\lambda}_i^{k+1} = \boldsymbol{\lambda}_i^k + \mu(\mathbf{P}_i\mathbf{x}^{k+1}-\mathbf{z}_i^{k+1}). \quad (13c)$$

Note that the minimization in (13a) is separable in $\mathbf{z}_i$, the $\mathbf{z}$-update rule can be expressed as

$$\mathbf{z}_i^{k+1} = \arg\min_{\mathbf{z}_i\in\mathcal{PP}_{d_i}} \frac{\mu}{2}\|\mathbf{z}_i\|^2 - (\mu\mathbf{P}_i\mathbf{x}^k+\boldsymbol{\lambda}_i^k)^T\mathbf{z}_i. \quad (14)$$

Thus we can derive the explicit form of the $\mathbf{z}$-update rule as

$$\mathbf{z}_i^{k+1} = \Pi_{\mathcal{PP}_{d_i}}\left(\mathbf{P}_i\mathbf{x}^k + \frac{\boldsymbol{\lambda}_i^k}{\mu}\right). \quad (15)$$

We now simplify the $\mathbf{x}$-update rule. It can be seen that $\mathbf{P}_i^T\mathbf{P}_i = \mathrm{Diag}(\mathbf{h}_i)$ due to the property of the selection matrix. By denoting $\mathbf{w} = \sum_i \mathbf{h}_i$, i.e., the j-th coordinate w_j equals the number of the check nodes connected with variable j, the augmented Lagrangian function can be rewritten as

$$L(\mathbf{x},\mathbf{z},\boldsymbol{\lambda}) = \frac{\mu}{2}\mathbf{x}^T\,\mathrm{Diag}(\mathbf{w})\mathbf{x} + (\boldsymbol{\gamma}+\sum_i \mathbf{P}_i^T(\boldsymbol{\lambda}_i-\mu\mathbf{z}_i))^T\mathbf{x} - \rho f(\mathbf{x}-\hat{\mathbf{x}}) + \sum\nolimits_i\left(\frac{\mu}{2}\|\mathbf{z}_i\|_2^2 - \boldsymbol{\lambda}_i^T\mathbf{z}_i\right). \quad (16)$$

For simplicity, we introduce the following notations:

$$\mathbf{D} = \mu\,\mathrm{Diag}(\mathbf{w}), \quad \mathbf{t}^k = -\boldsymbol{\gamma} - \sum\nolimits_i \mathbf{P}_i^T(\boldsymbol{\lambda}_i^k - \mu\mathbf{z}_i^{k+1}). \quad (17)$$

Then the $\mathbf{x}$-update rule can be expressed as

$$\mathbf{x}^{k+1} = \arg\min_{\mathbf{x}\in[0,1]^n} \frac{1}{2}\mathbf{x}^T\mathbf{D}\mathbf{x} - \mathbf{x}^T\mathbf{t}^k - \rho f(\mathbf{x}-\hat{\mathbf{x}}). \quad (18)$$

The explicit forms of the $\mathbf{x}$-update formula for particular choices of penalty term are given in Section III-B.

The ADMM algorithm for solving (7) is summarized in Algorithm 1.

Algorithm 1 ADMM Algorithm

Input: ρ, $\hat{\mathbf{x}}$.
Output: local solution $\mathbf{x}$.

1: **repeat**
2: Initialize $\mathbf{x}^0$ and $\boldsymbol{\lambda}^0$. Set $k=0$.
3: Update $\mathbf{z}$ by (15).
4: Update $\mathbf{x}$ by (18).
5: Update $\boldsymbol{\lambda}$ by (13c).
6: **until** some convergence criteria have been satisfied.

B. Examples of Penalty Functions

In the following, we focus on two examples of penalty functions.

1) ℓ_1-Norm: $f^{(1)}(\mathbf{x}) = \|\mathbf{x}\|_1$. In this case (12) becomes

$$L(\mathbf{x},\mathbf{z},\boldsymbol{\lambda}) = \boldsymbol{\gamma}^T\mathbf{x} - \rho\|\mathbf{x}-\hat{\mathbf{x}}\|_1 + \sum\nolimits_i \boldsymbol{\lambda}_i^T(\mathbf{P}_i\mathbf{x}-\mathbf{z}_i) + \frac{\mu}{2}\sum\nolimits_i \|\mathbf{P}_i\mathbf{x}-\mathbf{z}_i\|_2^2. \quad (19)$$

By applying (18) to (19), we derive the $\mathbf{x}$-update rule for this case as

$$\mathbf{x}^{k+1} = \Pi_{[0,1]^n}\left[\mathbf{D}^{-1}\left(\mathbf{t}^k - \rho\,\mathrm{sign}(\mathbf{D}\hat{\mathbf{x}}-\mathbf{t}^k)\right)\right]. \quad (20)$$

2) ℓ_2-Norm-Square: $f^{(2)}(\mathbf{x}) = \|\mathbf{x}\|_2^2$. Then (12) becomes

$$L(\mathbf{x},\mathbf{z},\boldsymbol{\lambda}) = \boldsymbol{\gamma}^T\mathbf{x} - \rho\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2 + \sum\nolimits_i \boldsymbol{\lambda}_i^T(\mathbf{P}_i\mathbf{x}-\mathbf{z}_i) + \frac{\mu}{2}\sum\nolimits_i \|\mathbf{P}_i\mathbf{x}-\mathbf{z}_i\|_2^2. \quad (21)$$

In this case, the $\mathbf{x}$-update rule can be expressed as

$$\mathbf{x}^{k+1} = \Pi_{[0,1]^n}[(\mathbf{D}-2\rho I)^{-1}\left(\mathbf{t}^k - 2\rho\hat{\mathbf{x}}\right)]. \quad (22)$$

For irregular codes, we can consider a weighted penalty term assigning varied parameters to variables with different degrees. The penalty function f can be selected as: (a) $f^{(3)}(\mathbf{x}) = \sum_j \kappa_j|x_j|$; (b) $f^{(4)}(\mathbf{x}) = \sum_j \kappa_j x_j^2$, corresponding to the ℓ_1 and ℓ_2 penalty respectively. Set W as the total number of the non-zero elements in $\mathbf{H}$. Then the weighted vector κ can be selected as

$$\kappa_j = n\frac{w_j}{W}. \quad (23)$$

Then the $\mathbf{x}$-update rule for penalty functions (a) and (b) can be expressed in a coordinate-wise form:

- weighted ℓ_1 norm:

$$x_j^{k+1} = \Pi_{[0,1]}\left[\frac{1}{\mu\kappa_j}(t_j^k - \rho\kappa_j\,\mathrm{sign}(w_j\hat{x}_j)\right]; \quad (24)$$

- weighted ℓ_2 norm:

$$x_j^{k+1} = \Pi_{[0,1]}\left[\frac{1}{\mu w_j - 2\rho\kappa_j}(t_j^k - 2\rho\kappa_j\hat{x}_j)\right]. \quad (25)$$

C. Restartable ADMM Algorithm

We now present our restartable ADMM decoding framework. In our proposed restartable decoding algorithm, $\hat{\mathbf{x}}$ is initialized as $\boldsymbol{\zeta}$, which denotes the vector whose entries are all 0.5. The coefficient ρ is initialized as $\rho^0 = \alpha+\beta^0$, where α and β^0 are two non-negative parameters. Whenever Algorithm 1 stops, we update $\hat{\mathbf{x}}$ and ρ according to the following rules and restart Algorithm 1. To be specific, in the N-th restart iteration, the parameter β is updated as $\beta^N = \xi\beta^{N-1}$, where $\xi\in(0,1)$ is an attenuation coefficient. Then the coefficient ρ is updated by

$$\rho^N = \alpha + \beta^N. \quad (26)$$

Denote $\mathbf{x}^l$ as the output of the l-th restart of Algorithm 1, we calculate the average of the historical outputs

$\bar{\mathbf{x}}^N := \frac{1}{N-1}\sum_{l=0}^{N-1}\mathbf{x}^l$. Then $\hat{\mathbf{x}}^N$ is set as a convex combination of ζ and $\bar{\mathbf{x}}^N$, i.e.,

$$\hat{\mathbf{x}}^N = \frac{\alpha}{\alpha+\beta^N}\boldsymbol{\zeta} + \frac{\beta^N}{\alpha+\beta^N}\bar{\mathbf{x}}^N. \tag{27}$$

Now we present the restartable decoding algorithm in Algorithm 2.

Algorithm 2 Restartable ADMM Decoding Algorithm

Input: log-likelihood ratio γ.
Output: $\hat{\mathbf{c}}$.

1: Initialization: Initialize $\hat{\mathbf{x}}^0 = \boldsymbol{\zeta}$, parameters α, β^0, β_L and $\xi \in (0,1)$. Compute $\rho^0 = \alpha + \beta^0$. Set $N = 0$ and T.
2: **repeat**
3: $\mathbf{x}^N \leftarrow$ ADMM$(\rho^N, \hat{\mathbf{x}}^N)$ using a relaxed convergence criterion in Remark 1.
4: $N \leftarrow N + 1$.
5: Compute $\beta^N \leftarrow \max(\xi\beta^{N-1}, \beta_L)$.
6: Compute $\rho^N \leftarrow \alpha + \beta^N$.
7: Compute $S \leftarrow \max(N, T)$, $L \leftarrow N - S$.
8: Compute $\bar{\mathbf{x}}^N \leftarrow \frac{1}{S}\sum_{l=L}^{N-1}\mathbf{x}^l$.
9: Compute $\hat{\mathbf{x}}^N$ according to (27).
10: Compute $\hat{\mathbf{c}} \leftarrow \text{round}(\mathbf{x}^N)$.
11: **until** $\mathbf{H}\otimes\hat{\mathbf{c}} = 0$ or the maximal iteration number has been reached.

In each iteration of Algorithm 2, the ADMM framework is applied to solve the problem (7). Whenever the ADMM algorithm converges to a stationary point, the negative proximal term will be updated and the ADMM solver will be restarted. Since $\hat{\mathbf{x}}$ is an average of 0.5 and all the local solutions that have been reached during the decoding process, the negative proximal term will push the decoder to find new solutions.

Remark 1: (i) To accelerate the ADMM iterations, we set a relaxed convergence criterion for Algorithm 1. We calculate the nearest binary vector to $\mathbf{x}^k$ in each iteration, and stop the ADMM process whenever this vector remains the same after one iteration, i.e., round$(\mathbf{x}^k)$ = round$(\mathbf{x}^{k+1})$. Thus the ADMM algorithm can stop in a few iterations and output a local solution. (ii) To reduce the memory usage and computational complexity, we set $\hat{\mathbf{x}}$ as the average of at most the last T solutions that the ADMM algorithm has generated instead of all the historical ones. A lower bound β_L is set to avoid the parameter β from vanishing. (iii) Although our algorithm needs some extra computation derived from the updates of $\hat{x}, \alpha, \beta$, it still has similar computational complexity as other ADMM-based algorithms, since the main computational complexities are from the updates of the variables x, z, λ in the ADMM steps which need multiplication of matrices and vectors, and the extra computation of $\hat{x}, \alpha, \beta$ only needs the multiplication of scalars and the addition of vectors whose complexities are significantly lower than the multiplication of matrices and vectors. Besides, the extra variables $\hat{x}, \alpha, \beta$ are only updated in the outer iterations, while the updates of the variables x, z, λ are conducted in each inner iteration.

IV. Performance Analysis

In this section, we make an analysis of Algorithms 1 and 2. We consider two aspects of our proposed algorithm, including convergence property and decoding performance analysis.

A. Convergence Property

We consider the case that $f(\mathbf{x}) = \|\mathbf{x}\|_2^2$ and define $r(\mathbf{x}) := \boldsymbol{\gamma}^T\mathbf{x} - \rho f(\mathbf{x} - \hat{\mathbf{x}})$. Obviously, $r(\mathbf{x})$ is Lipschitz differentiable with constant 2ρ. We let $\delta_Z(\cdot)$ denote the indicator function of the set Z, i.e., $\delta_Z(\mathbf{z}) = \begin{cases} +\infty, & \text{if } \mathbf{z} \in Z, \\ 0, & \text{if } \mathbf{z} \notin Z. \end{cases}$ Define the matrix $\mathbb{P} := [\mathbf{P}_1; \mathbf{P}_2; \ldots; \mathbf{P}_m]$ and observe that

$$\mathbb{P}^T\mathbb{P} = \sum\nolimits_i \mathbf{P}_i^T\mathbf{P}_i = \text{Diag}(\mathbf{w}).$$

We denote τ by the smallest element of $\mathbf{w}$. It is easy to see that $\tau > 0$.

Definition 1: $(\mathbf{x}^*, \mathbf{z}^*, \lambda^*)$ is a stationary point of the problem (7) if it satisfies the first-order necessary optimality condition, that is,

$$\begin{cases} 0 = \nabla r(\mathbf{x}^*) + \sum_i \mathbf{P}_i^T\lambda_i{}^*, \\ 0 = \mathbf{P}_i\mathbf{x}^* - \mathbf{z}_i^*, \quad \forall\ i \in \mathcal{I}, \\ \lambda_i^* \in \partial\delta_{\mathcal{P}\mathcal{P}_{d_i}}(\mathbf{z}_i^*), \quad \forall\ i \in \mathcal{I}. \end{cases} \tag{28}$$

Our main goal is to find conditions that ensure the convergence of the proposed algorithm. We split the proof of the main result into several lemmas.

Lemma 1: It holds that for any $k = 1, 2, \ldots$,

$$\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\| \le \frac{2\rho}{\sqrt{\tau}}\|\mathbf{x}^{k+1} - \mathbf{x}^k\|. \tag{29}$$

Proof: It follows from the optimality condition of $\mathbf{x}^k$ that

$$\nabla r(\mathbf{x}^k) + \sum\nolimits_i \mathbf{P}_i^T\boldsymbol{\lambda}_i^{k-1} + \mu\sum\nolimits_i \mathbf{P}_i^T(\mathbf{P}_i\mathbf{x}^k - \mathbf{z}_i^k) = 0,$$

which together with the iteration of $\boldsymbol{\lambda}^k$ in (13c) yields that

$$-\nabla r(\mathbf{x}^k) = \sum\nolimits_i \mathbf{P}_i^T\boldsymbol{\lambda}_i^k = \mathbb{P}^T\boldsymbol{\lambda}^k. \tag{30}$$

Note that the image of selection matrix $\mathbf{P}_i$ satisfies $\text{Im}(\mathbf{I}) = \text{Im}(\mathbf{P}_i) = \mathbb{R}^{d_i}$, which implies that $\boldsymbol{\lambda}_i^{k+1} - \boldsymbol{\lambda}_i^k = \mu(\mathbf{P}_i\mathbf{x}^{k+1} - \mathbf{z}_i^{k+1}) \in \text{Im}(\mathbf{P}_i)$ for every $i \in \mathcal{I}$. Then $\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k \in \text{Im}(\mathbb{P})$, and thus there exists a vector v such that

$$\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k = \mathbb{P}v. \tag{31}$$

It gives us

$$\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|^2 = \|\mathbb{P}v\|^2 \ge \lambda_{\min}(\mathbb{P}^T\mathbb{P})\|v\|^2 = \tau\|v\|^2,$$

which implies that $\|v\| \le \frac{1}{\sqrt{\tau}}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|$. This together with (31) yields that

$$\begin{aligned}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|^2 &= (\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k)^T\mathbb{P}v \le \|\mathbb{P}^T(\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k)\|\|v\| \\ &\le \frac{1}{\sqrt{\tau}}\|\mathbb{P}^T(\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k)\|\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|.\end{aligned}$$

Therefore,

$$\begin{aligned}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\| &\le \frac{1}{\sqrt{\tau}}\|\mathbb{P}^T(\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k)\| \\ &\overset{(30)}{=} \frac{1}{\sqrt{\tau}}\|\nabla r(\mathbf{x}^{k+1}) - \nabla r(\mathbf{x}^k)\| \\ &= \frac{2\rho}{\sqrt{\tau}}\|\mathbf{x}^{k+1} - \mathbf{x}^k\|,\end{aligned}$$

which completes the proof. ■

The next lemma ensures the descent property of the proposed algorithm.

Lemma 2: If $\mu \geq (4\rho+2)/\tau$, there is a constant $C_1 > 0$ such that for any $k = 1, 2, \ldots$,

$$L(\mathbf{x}^k, \mathbf{z}^k, \boldsymbol{\lambda}^k) - L(\mathbf{x}^{k+1}, \mathbf{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) \geq C_1 \left(\|\mathbf{x}^k - \mathbf{x}^{k+1}\|^2 + \|\mathbf{z}^k - \mathbf{z}^{k+1}\|^2 \right). \quad (32)$$

Proof: We first prove

$$L(\mathbf{x}^k, \mathbf{z}^k, \boldsymbol{\lambda}^k) - L(\mathbf{x}^k, \mathbf{z}^{k+1}, \boldsymbol{\lambda}^k) \geq \frac{\mu}{2}\|\mathbf{z}^k - \mathbf{z}^{k+1}\|^2. \quad (33)$$

The optimality of $\mathbf{z}^{k+1}$ implies that for any $i \in \mathcal{I}$,

$$\boldsymbol{\lambda}_i^k + \mu(\mathbf{P}_i\mathbf{x}^k - \mathbf{z}_i^{k+1}) \in \partial \delta_{\mathcal{PP}_{d_i}}(\mathbf{z}_i^{k+1}).$$

Since the sub-differential of the indicator function at $\mathbf{z}_i^{k+1}$ is known as the normal cone of $\mathcal{PP}_{d_i}$, it can be seen directly from its definition that

$$\left(\boldsymbol{\lambda}_i^k + \mu(\mathbf{P}_i\mathbf{x}^k - \mathbf{z}_i^{k+1})\right)^T \left(\mathbf{z}_i^k - \mathbf{z}_i^{k+1}\right) \leq 0.$$

Then for $\mathbf{z}_i^k, \mathbf{z}_i^{k+1} \in \mathcal{PP}_{d_i}$, we have the estimate

$$\begin{aligned} &L(\mathbf{x}^k, \mathbf{z}^k, \boldsymbol{\lambda}^k) - L(\mathbf{x}^k, \mathbf{z}^{k+1}, \boldsymbol{\lambda}^k) \\ &= -\sum_i \left(\boldsymbol{\lambda}_i^k + \mu(\mathbf{P}_i\mathbf{x}^k - \mathbf{z}_i^{k+1})\right)^T \left(\mathbf{z}_i^k - \mathbf{z}_i^{k+1}\right) \\ &\quad + \frac{\mu}{2}\|\mathbf{z}^k - \mathbf{z}^{k+1}\|^2 \geq \frac{\mu}{2}\|\mathbf{z}^k - \mathbf{z}^{k+1}\|^2. \end{aligned}$$

We next prove

$$L(\mathbf{x}^k, \mathbf{z}^{k+1}, \boldsymbol{\lambda}^k) - L(\mathbf{x}^{k+1}, \mathbf{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) \geq \|\mathbf{x}^k - \mathbf{x}^{k+1}\|^2.$$

Since the matrix $\mathbb{P}$ has full column rank, we can obtain

$$\begin{aligned} &L(\mathbf{x}^k, \mathbf{z}^{k+1}, \boldsymbol{\lambda}^k) - L(\mathbf{x}^{k+1}, \mathbf{z}^{k+1}, \boldsymbol{\lambda}^{k+1}) \\ &= r(\mathbf{x}^k) - r(\mathbf{x}^{k+1}) + \sum_i (\boldsymbol{\lambda}_i^{k+1})^T(\mathbf{P}_i\mathbf{x}^k - \mathbf{P}_i\mathbf{x}^{k+1}) \\ &\quad + \frac{\mu}{2}\sum_i \|\mathbf{P}_i\mathbf{x}^k - \mathbf{P}_i\mathbf{x}^{k+1}\|^2 - \frac{1}{\mu}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|^2 \\ &\overset{(30)}{=} r(\mathbf{x}^k) - r(\mathbf{x}^{k+1}) - \langle \nabla r(\mathbf{x}^{k+1}), \mathbf{x}^k - \mathbf{x}^{k+1}\rangle \\ &\quad + \frac{\mu}{2}\|\mathbb{P}\mathbf{x}^k - \mathbb{P}\mathbf{x}^{k+1}\|^2 - \frac{1}{\mu}\|\boldsymbol{\lambda}^{k+1} - \boldsymbol{\lambda}^k\|^2 \\ &\overset{(29)}{\geq} -\rho\|\mathbf{x}^k - \mathbf{x}^{k+1}\|^2 + \frac{\mu\tau}{2}\|\mathbf{x}^k - \mathbf{x}^{k+1}\|^2 \\ &\quad - \frac{4\rho^2}{\tau\mu}\|\mathbf{x}^k - \mathbf{x}^{k+1}\|^2 \geq \|\mathbf{x}^k - \mathbf{x}^{k+1}\|^2, \end{aligned} \quad (34)$$

where the last inequality holds because

$$-\rho + \frac{\mu\tau}{2} - \frac{4\rho^2}{\tau\mu} \geq -\rho + 2\rho + 1 - \frac{\rho^2}{2\rho+1} \geq 1.$$

Summing up (34) and (33) yields (32), where $C_1 := \min\{-\rho + \frac{\mu\tau}{2} - \frac{4\rho^2}{\tau\mu}, \frac{\mu}{2}\}$. ∎

Now we are ready to prove the convergence property of the proposed ADMM algorithm.

Theorem 2: The sequence $\{(\mathbf{x}^k, \mathbf{z}^k, \lambda^k)\}$ generated by Algorithm 1 converges globally to the unique limit point $(\mathbf{x}^*, \mathbf{z}^*, \lambda^*)$ for any sufficiently large $\mu \geq (4\rho+2)/\tau$, and $(\mathbf{x}^*, \mathbf{z}^*, \lambda^*)$ is a stationary point of the problem (7).

Proof: It suffices to verify the conditions in [24] for ADMM to converge to a stationary point. Observe that the model (7) can be equivalently written as

$$\begin{aligned} &\min\ r(\mathbf{x}) + \sum_i \delta_{\mathcal{PP}_{d_i}}(\mathbf{z}_i), \\ &\text{s.t. } \mathbf{P}_i\mathbf{x} = \mathbf{z_i}, \quad \forall\ i \in \mathcal{I}. \end{aligned} \quad (35)$$

We can observe that $\delta_{\mathcal{PP}_{d_i}}(\mathbf{z}_i)$ is a prox-regular function since the set $\mathcal{PP}_{d_i}$ is the convex hull of all the binary vectors. The matrix $\mathbf{P}_i$ satisfies $\mathrm{Im}(\mathbf{I}) \subseteq \mathrm{Im}(\mathbf{P}_i)$ for every $i \in \mathcal{I}$. We can deduce from Lemma 2 that the sequence $\{(\mathbf{x}^k, \mathbf{z}^k)\}$ generated by the proposed algorithm is bounded. Then the conditions from Theorem 1 in [24] hold for the problem (7). Hence the sequence generated by Algorithm 1 is bounded and it has at least one limit point.

Furthermore, we can easily verify that $L(\mathbf{x}, \mathbf{z}, \boldsymbol{\lambda}) + \sum_i \delta_{\mathcal{PP}_{d_i}}(\mathbf{z}_i)$ is a Kurdyka-Łojasiewicz (KŁ) function [25]. Then $(\mathbf{x}^k, \mathbf{z}^k, \lambda^k)$ converges globally to the unique limit point $(\mathbf{x}^*, \mathbf{z}^*, \lambda^*)$, which is a stationary point. ∎

Remark 2: Benefiting from the special structure of the model (7), we show the convergence properties of Algorithm 1 even though the problem is nonconvex. Since the objective function of our decoding model changed in each restart, it is not straightforward to construct global convergence results for Algorithm 2. However, in our simulation result, one can find that Algorithm 2 can converge in most cases.

B. Decoding Performance Analysis

1) ML Test: Our proposed Algorithm 2 has the following ML certificate properties.

Proposition 1: If the output from Algorithm 2 is a feasible solution of (7) and is also integral, then it is a valid codeword.

Proof: This follows from all the feasible integral solutions of (7) are valid codewords. ∎

Proposition 2: If the output from Algorithm 2 is a global minimizer of (7) and is also integral, and the corresponding ρ satisfies the conditions in Theorem 1, then it is an ML solution.

Proof: This follows directly from Theorem 1. ∎

In general, we cannot determine whether the output of Algorithm 2 is a global solution of (7). However, Algorithm 2 has an ML certificate property similar to [18], where we can test whether the integral output is the ML solution or not. This test is shown as follows.

Proposition 3: If the output of Algorithm 2 is an integral solution, we do one more restart by setting the parameter ρ to zero. If the new output obtained in this way maintains the same, then this solution is an ML solution.

Proof: By setting the parameter ρ to zero, we obtain the LP decoding problem. Since the ADMM algorithm for the LP problem has the guarantee to converge to a global minimum, the output of Algorithm 2 is a global solution of (4). Therefore, it is the ML solution by the ML certificate of LP decoding. ∎

2) All-Zero Assumptions: Our proposed decoder satisfies the all-zero assumption. Thus we can assume the codeword transmitted by the channel is all-zero without loss of generality. This property is presented formally as follows.

Theorem 3: The failure probability of Algorithm 2 is independent of the transmitted codeword.

TABLE I
PARAMETER SELECTION

Code	β^0	β_L	α	Code	β^0	β_L	α
$\mathcal{C}_1$	0.7	0.46	1.6	$\mathcal{C}_4$	0.1	0.02	1.6
$\mathcal{C}_2$	0.7	0.36	1.2	$\mathcal{C}_5$	0.4	0.1	1.2
$\mathcal{C}_3$	0.3	0.14	1.7	$\mathcal{C}_6$	0.4	0.12	1.3

Proof: We give a sketch of proof here, and the details are shown in the appendix. Let $\mathbf{c}$ denote any codeword, and $\mathcal{B}(\mathbf{c})$ denote the set of outputs of the channel that make the decoder fail to decode $\mathbf{c}$. We can prove that a one-to-one mapping between $\mathcal{B}(\mathbf{c})$ and $\mathcal{B}(\mathbf{0})$ exists. Therefore the failure probability for transmitting any codeword equals the failure probability for transmitting the all-zero codeword. ■

V. SIMULATION RESULTS

In this section, several simulation results are presented to show the effectiveness of the restartable ADMM decoder. In Section V-A, we compare the decoding FER performance and decoding efficiency of the restartable ADMM decoder using both ℓ_1 and ℓ_2-square penalties with the min-sum decoder, the sum-product decoder and other ADMM-based decoders. In section V-B, we show the choice of parameters of our proposed decoder.

A. Performances of the Proposed Algorithm

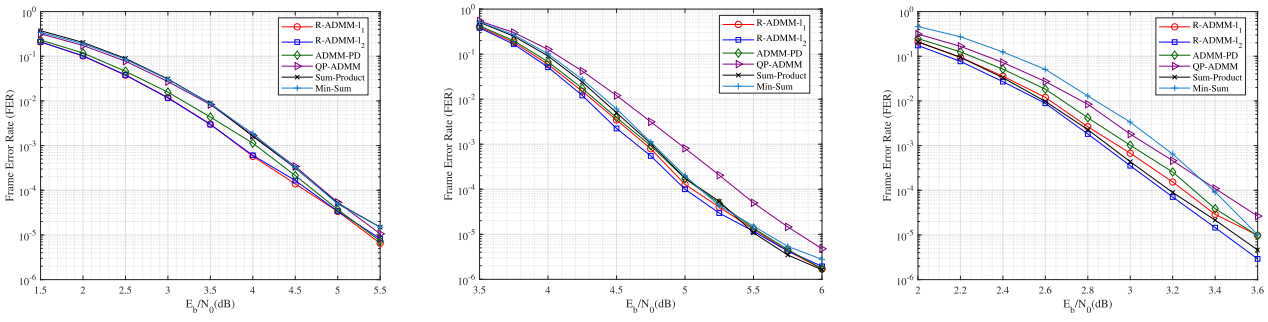
In this subsection, we present simulation results for six binary LDPC codes. We choose three regular codes, denoted by $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$, which are $(96, 48)$ MacKay 96.33.964 LDPC code, rate-0.89 MacKay $(999, 888)$ LDPC code, and WiFi $(648, 432)$ LDPC code, respectively. We also test three irregular LDPC codes, denoted by $\mathcal{C}_4$, $\mathcal{C}_5$ and $\mathcal{C}_6$, which are 5G-LDPC code with base graph 1 and $Z_c = 16$, rate-0.5 WiMax $(576, 288)$ LDPC code, and rate-0.75 WiMax $(1152, 864)$ LDPC code. The considered decoders include QP-ADMM decoder [17], ADMM-based penalized decoder (ADMM-PD) [18], and the classical sum-product decoder.

The parameters for our proposed decoder are set as follows: μ is chosen to be 3 for the ℓ_1 penalty and 4 for the ℓ_2-square penalty. The rest parameters of the ℓ_2-square penalty for six binary LDPC codes are set according to Table I. For the irregular LDPC codes $\mathcal{C}_4$, $\mathcal{C}_5$ and $\mathcal{C}_6$, we apply the weighted form penalty function claimed in Section III-B, where the parameter κ is chosen according to (23). The parameters for other ADMM-based decoders are chosen based on a grid search. We stop iterations when the early termination scheme $[\mathbf{Hx}]_2 = 0$ is satisfied or the maximum number of iterations is reached. We set the maximum iteration number of all the decoders as 500. Since our proposed decoding algorithm contains inner iterations (ADMM updates) and outer iterations (restarts), we consider the number of total iterations when counting the iteration number of our proposed decoder. All the codewords are transmitted over the additive white Gaussian noise (AWGN) channel.

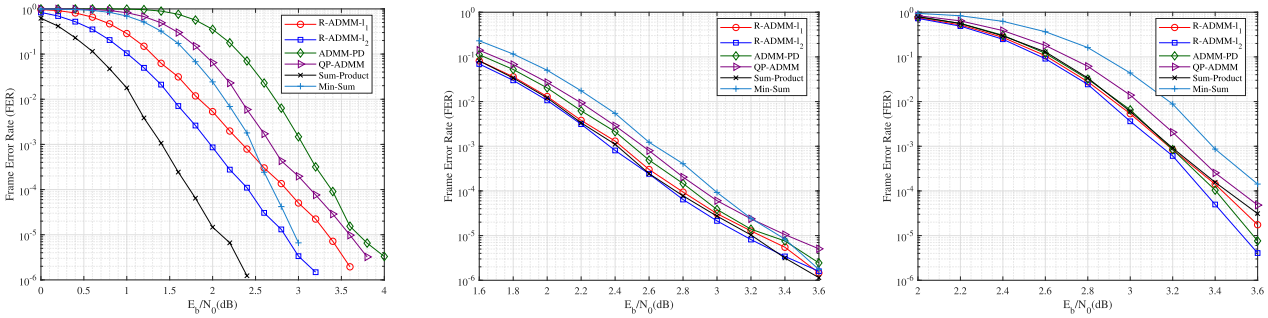
Fig. 2 shows the FER performance of the mentioned six LDPC codes. The points plotted in all the curves are based on collecting 100 frame errors except for the last two SNR values, where we generate 20 errors due to limited computing resources. We also impose a minimum number of simulated frames as 50000 to reduce the statistical error. From Fig. 2(a), one can see that our proposed decoder displays better FER performance than other considered decoders in low SNR regions. However, in high SNR regions, the gap between our proposed decoder and the ADMM penalized decoder [18] narrows. A similar phenomenon can also be observed in Fig. 2(b). The sum-product decoder surpasses our proposed decoder at high SNRs. Fig. 2(c) shows that our proposed decoder using ℓ_2 penalty achieves better FER performance than other considered decoders at both low and high SNRs. In Fig. 2(d), it can be seen that our proposed decoder obtains a significant improvement in the FER performance compared with other ADMM-based decoders. This is because the degrees of variable nodes in 5G LDPC codes vary dramatically. The negative proximal terms with weighted forms can improve the error-correction performance in this case. Fig. 2(e) shows that our proposed decoder outperforms the other ADMM-based decoders and the min-sum decoder in a wide SNR region, and performs similarly to the sum-product decoder. Similarly to Fig. 2(a) and Fig. 2(b), the FER performance of our proposed decoder in Fig. 2(e) gets close to that of the ADMM penalized decoder in high SNR regions. In Fig. 2(f), one can observe that the restartable decoder using ℓ_2-square penalty has the best decoding performance. Our decoder using ℓ_1 penalty performs similarly to the ADMM penalized decoder and the sum-product decoder, and outperforms the QP-ADMM decoder [17] and the min-sum decoder. From Fig. 2, one can see that our proposed decoder attains better error-correction performance in most cases.

Figs. 3(a)-8(a) compare the average number of total iterations to output a codeword, i.e., a binary vector satisfies $\mathbf{Hx} = 0$, of $\mathcal{C}_1$-$\mathcal{C}_6$ between our proposed algorithm with other considered ADMM-based decoders. In cases where the algorithm cannot find a binary vector satisfying $\mathbf{Hx} = 0$, the number of total iterations will be counted as 500, i.e., the maximum number of iterations. From these figures, one can find that all of the compared decoders at low SNRs need more iterations to output a codeword. The reason is that in low SNR regions transmitted bits are affected by stronger noise, which makes it harder to recover the codeword and the algorithm often reaches the maximum number of iterations. Moreover, it can also be seen that our proposed decoder shares similar iteration numbers with the ADMM penalized decoder [18] in $\mathcal{C}_1$ and $\mathcal{C}_2$, and needs less number of iterations in $\mathcal{C}_3$-$\mathcal{C}_6$. QP-ADMM decoder needs more iterations to find a codeword in these cases. Thus, our proposed decoder achieves lower decoding complexity than other ADMM-based decoders in most cases. The min-sum decoder needs more iteration numbers than our proposed decoder in the low SNR region. However, it surpasses our proposed decoder at high SNRs. The sum-product decoder needs the least iteration number in many cases, but its computational complexity can be higher due to the calculation of the $tanh$ function in each iteration.

Figs. 3(b)-8(b) plot the average number of restarts for our proposed decoding algorithm to output a codeword using both

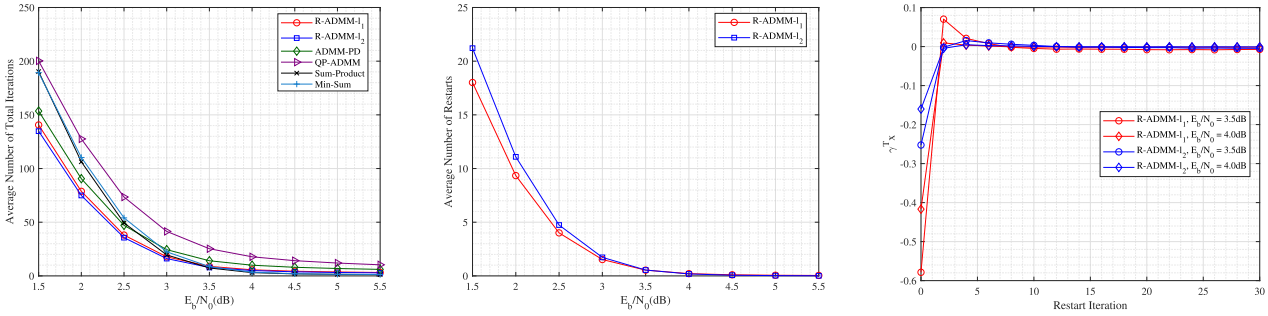


(a) $\mathcal{C}_1$-$(96, 48)$, FER. (b) $\mathcal{C}_2$-$(999, 888)$, FER. (c) $\mathcal{C}_3$-$(648, 432)$, FER.



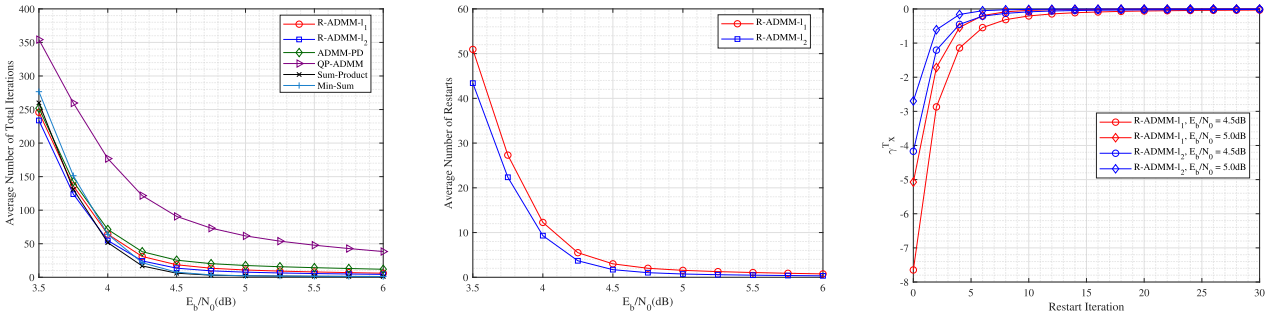
(d) $\mathcal{C}_4$-$(1088, 352)$, FER. (e) $\mathcal{C}_5$-$(576, 288)$, FER. (f) $\mathcal{C}_6$-$(1152, 864)$, FER.

Fig. 2. Comparisons of FER performance for six LDPC codes using different decoders, where "R-ADMM-l_1" and "R-ADMM-l_2" denote the restartable ADMM decoder using ℓ_1 and ℓ_2 penalties, respectively.



(a) Average number of total iterations versus SNR. (b) Average number of restarts versus SNR. (c) Objective function values versus restart iteration number.

Fig. 3. Numerical experiments on the convergence property of our decoder and other ADMM-based decoders for $\mathcal{C}_1$.

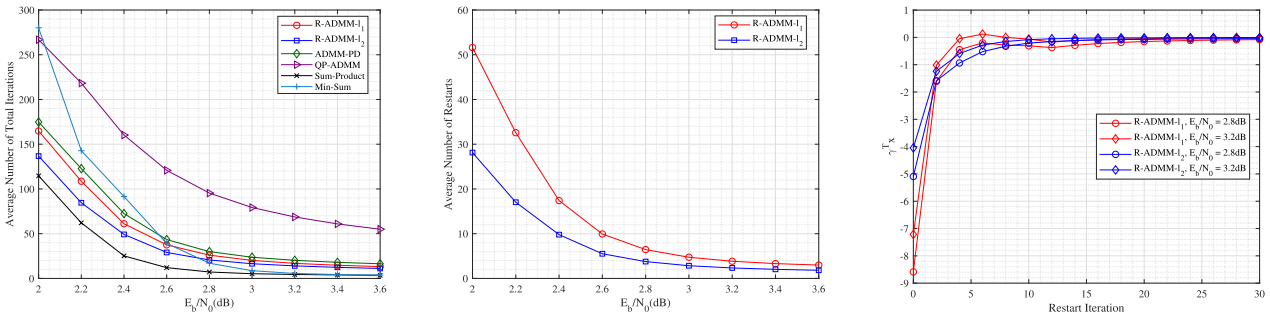


(a) Average number of total iterations versus SNR. (b) Average number of restarts versus SNR. (c) Objective function values versus restart iteration number.

Fig. 4. Numerical experiments on the convergence property of our decoder and other ADMM-based decoders for $\mathcal{C}_2$.

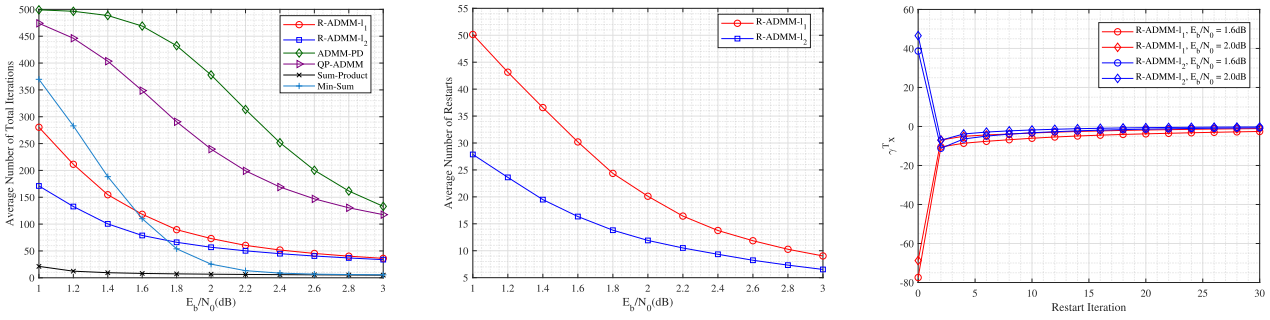
ℓ_1 and ℓ_2 penalty functions. It can be seen that the number of restarts decreases as the SNR increases. The reason is that in high SNR regions, the ADMM algorithm can easily find a codeword within a few iterations, and thus there is no need for our decoder to restart. These curves can also explain the phenomenon that the decoding performance of our proposed decoder and the ADMM penalized decoder [18] get close in high SNR regions.

Figs. 3(c)-8(c) show the convergence performance of our proposed decoder for codes $\mathcal{C}_1$-$\mathcal{C}_6$. The y-axis denotes the linear term of the objective function, and the x-axis denotes the restart iteration, i.e., the current number of restarts of our proposed decoder. All curves in Figs. 3(c)-8(c) are plotted based on one million LDPC frames. We use all-zero vectors as our transmitted code because our proposed decoder satisfies the all-zero assumptions. When our decoding framework



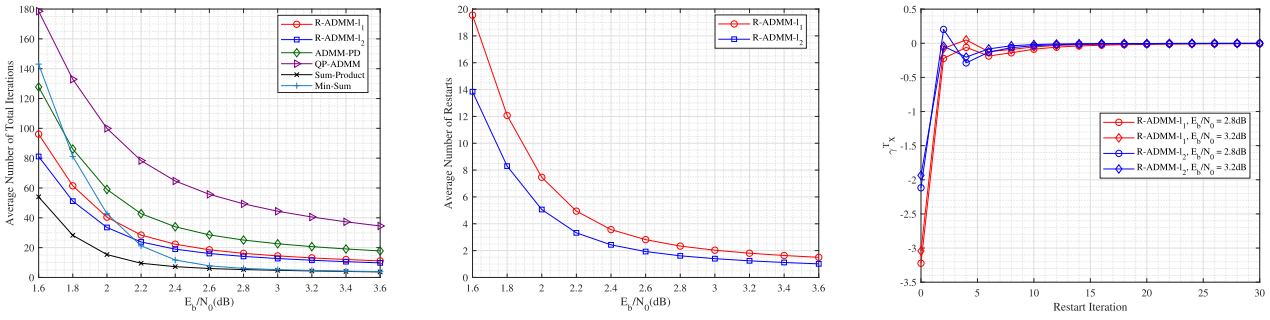
(a) Average number of total iterations versus SNR. (b) Average number of restarts versus SNR. (c) Objective function values versus restart iteration number.

Fig. 5. Numerical experiments on the convergence property of our decoder and other ADMM-based decoders for $\mathcal{C}_3$.



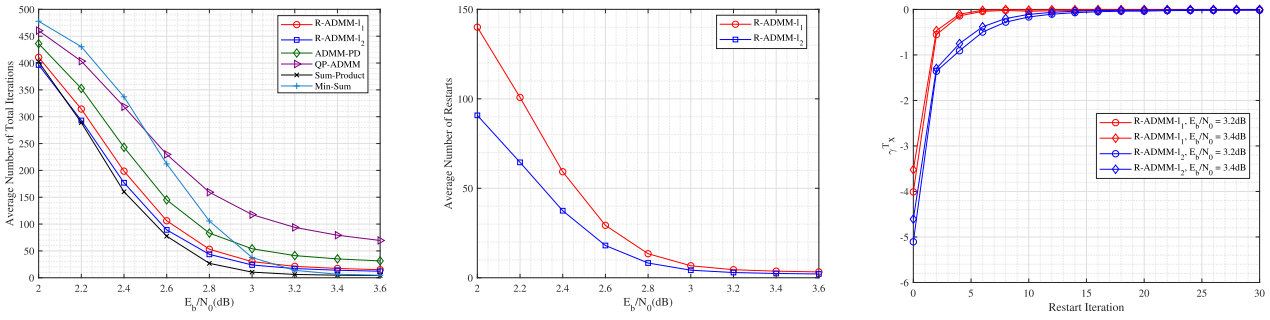
(a) Average number of total iterations versus SNR. (b) Average number of restarts versus SNR. (c) Objective function values versus restart iteration number.

Fig. 6. Numerical experiments on the convergence property of our decoder and other ADMM-based decoders for $\mathcal{C}_4$.



(a) Average number of total iterations versus SNR. (b) Average number of restarts versus SNR. (c) Objective function values versus restart iteration number.

Fig. 7. Numerical experiments on the convergence property of our decoder and other ADMM-based decoders for $\mathcal{C}_5$.



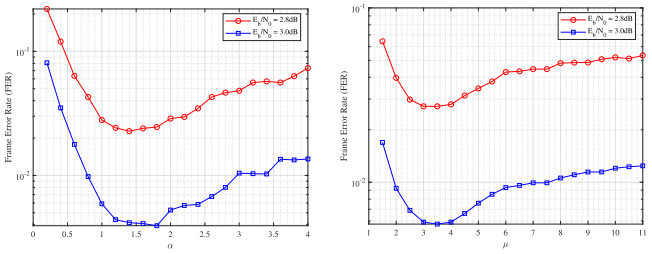
(a) Average number of total iterations versus SNR. (b) Average number of restarts versus SNR. (c) Objective function values versus restart iteration number.

Fig. 8. Numerical experiments on the convergence property of our decoder and other ADMM-based decoders for $\mathcal{C}_6$.

converges, the value of the linear term in our decoding model (7) should be zero. From these figures, one can find that our proposed decoding framework can converge within at most 30 times of restarts in most cases, though the negative proximal term is updated and the objective function of our model is changed after each restart.

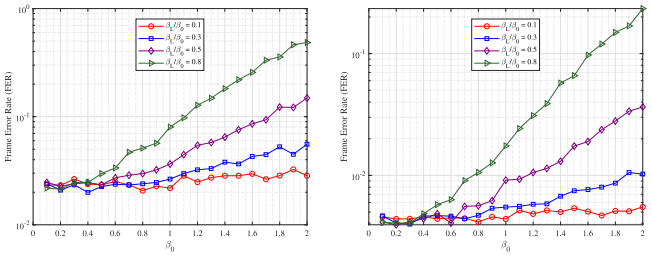
B. Choices of Parameters

The parameters in our proposed decoder can affect the decoding performance. In this subsection, we show several simulations based on different choices of these parameters. From the simulation results, we can find the suitable choice of parameters to obtain the best decoding performance.



(a) FER performance versus α. (b) FER performance versus μ.

Fig. 9. FER performance of the $(1152, 864)$ WiMax LDPC code plotted as a function of α or μ with different SNRs.



(a) FER performance versus β^0 with $E_b/N_0 = 2.8$dB. (b) FER performance versus β^0 with $E_b/N_0 = 3.0$dB.

Fig. 10. FER performance of the $(1152, 864)$ WiMax LDPC code plotted as a function of β^0 with different β^L/β^0.

In Fig. 9, we plot the FER performance of code $\mathcal{C}_6$ as the function of parameters α and μ at two different SNRs with other parameters fixed according to Table I. We collect 100 frame errors for each data point in these curves. From Fig. 9, one can find that the FER curves take on an initially downward and then upward trend for parameters α and μ. In comparison with the FER curve, we find that $\alpha \in [1.2, 1.8]$ and $\mu \in [3, 4]$ lead to better decoding performance.

In Fig. 10, we investigate the effects of the parameters β^0 and β_L. Since β_L should never be greater than β^0, we plot the FER performance as the function of β^0 with fixed β^L/β^0 lower than 1. We consider two different SNR levels in Fig. 10, and each curve is plotted based on 100 frame errors. From Fig. 10, one can find that $\beta^0 \in [0.1, 0.5]$ and $\beta^L = 0.3\beta^0$ can be a suitable choice for our decoder in this case.

VI. Conclusion

In this paper, we propose a restartable ADMM-based decoding algorithm. By introducing a negative proximal term to the LP decoding problem, we present a restartable decoding model and use the ADMM algorithm to solve it. Whenever the ADMM algorithm converges, the negative proximal term will be updated and the ADMM algorithm will be restarted. The negative proximal term penalizes all the stationary points that have been found during the decoding process, and pushes the algorithm to find new local minimums. Therefore plenty of candidate solutions will be found during one decoding process, which can raise the possibility to obtain a codeword that satisfies the parity-check equation. We propose some properties the negative proximal term should have, and investigate several choices of the negative proximal term. We also discuss the convergence property of our proposed decoder, and prove that the decoding error possibility is independent of the transmitted codeword. We implement numerical experiments which demonstrate that our proposed decoder outperforms other ADMM-based in most cases and obtains similar decoding complexity.

Appendix
Proof of Theorem 3

In [18], the authors established this property for ADMM penalized decoder which solved the problem (6). We now prove Theorem 3 following the path in [18]. Denote the set of all the possible outputs of the channel when transmitting codeword $\mathbf{c}$ by $\mathcal{A}(\mathbf{c})$, then our target theorem is a straightforward consequence of the following lemma:

Lemma 3: There exists a one-to-one mapping $Z : \mathbf{y} \mapsto \mathbf{y}^0$ from $\mathcal{A}(\mathbf{c})$ to $\mathcal{A}(\mathbf{0})$ that satisfies

1) $\Pr(\mathbf{y}|\mathbf{c}) = \Pr(\mathbf{y}^0|\mathbf{0})$.
2) $\mathbf{y} \in \mathcal{B}(\mathbf{c})$ *if and only if* $\mathbf{y}^0 \in \mathcal{B}(\mathbf{0})$.

Proof: The framework of our proof is the same as in [18]. We define the mapping $Z : \mathbf{y} \mapsto \mathbf{y}^0$ as

$$y_i^0 = \begin{cases} y_i, & \text{if } c_i = 0, \\ y_i', & \text{if } c_i = 1, \end{cases}$$

where y_i' is the symmetric symbol of y_i with respect to the channel. To complete our proof, we only need to verify that Lemmas 18 and 19 in [18] also hold for our proposed decoder. Therefore, we prove the following lemmas, where Lemmas 4 and 5 correspond to Lemma 18 in [18], and Lemma 7 corresponds to Lemma 19 in [18]. ■

We use the same definitions for operators $R_{\mathbf{c}}$ and $T_{\mathbf{c}}$ with respect to a codeword $\mathbf{c}$ as in [18]:

$$R_{\mathbf{c}}(\mathbf{a})_i = \begin{cases} a_i, & \text{if } c_i = 0, \\ 1 - a_i, & \text{if } c_i = 1, \end{cases} \quad T_{\mathbf{c}}(\mathbf{a})_i = R_{\mathbf{c}}(\mathbf{a})_i + c_i.$$

We represent $R_{\mathbf{c}}(\mathbf{z}_i)$ by taking the same operations to $\mathbf{z}_i$ with respect to the corresponding sub-vector of $\mathbf{c}$. Then we have the following lemmas which claim that there exists a one-to-one mapping between the iteration sequences of decoding any codeword $\mathbf{c}$ and the all-zero codeword.

Lemma 4: Suppose that $\mathbf{z}_i$, $\mathbf{x}$ and $\boldsymbol{\lambda}_i$ are the variables in the k-th iteration of Algorithm 1 when decoding $\mathbf{y}$ with input ρ and $\hat{\mathbf{x}}$. $\mathbf{z}_i^+$, $\mathbf{x}^+$ and $\boldsymbol{\lambda}_i^+$ are the $(k+1)$-th iteration of Algorithm 1. Meanwhile, suppose that $\mathbf{z}_i^0$, $\mathbf{x}^0$ and $\boldsymbol{\lambda}_i^0$ are the variables in the k-th iteration of Algorithm 1 when decoding $\mathbf{y}^0$ with input ρ and $\hat{\mathbf{x}}^0$. $\mathbf{z}_i^{0,+}$, $\mathbf{x}^{0,+}$ and $\boldsymbol{\lambda}_i^{0,+}$ are the $(k+1)$-th iteration of Algorithm 1. If $\mathbf{z}_i = R_{\mathbf{c}}(\mathbf{z}_i^0)$, $\mathbf{x} = R_{\mathbf{c}}(\mathbf{x}^0)$, $\boldsymbol{\lambda}_i = T_c(\boldsymbol{\lambda}_i^0)$ and $\hat{\mathbf{x}} = R_{\mathbf{c}}(\hat{\mathbf{x}}^0)$, then we have $\mathbf{z}_i^+ = R_{\mathbf{c}}(\mathbf{z}_i^{0,+})$, $\mathbf{x}^+ = R_{\mathbf{c}}(\mathbf{x}^{0,+})$ and $\boldsymbol{\lambda}_i^+ = T_c(\boldsymbol{\lambda}_i^{0,+})$.

Proof: Let $\boldsymbol{\gamma}$ and $\boldsymbol{\gamma}^0$ denote the log-likelihood ratios corresponding to $\mathbf{y}$ and $\mathbf{y}^0$ respectively. Now we verify $\mathbf{x}^+ = R_{\mathbf{c}}(\mathbf{x}^{0,+})$. From the definition of $R_{\mathbf{c}}$, we only need to consider the indices j that satisfy $c_j = 1$. From Algorithm 1, x_j^+ is the optimal solution to the following equation that has the maximal distance with 0.5:

$$x_j^+ = \arg\min_{x_j \in [0,1]} \frac{1}{2}\mu w_j x_j^2 - t_j x_j - \rho f_j(x_j - \hat{x}_j). \tag{36}$$

Now we define u_j^+ as the root of the following equation:

$$\mu w_j x - t_j - \rho f_j'(x - \hat{x}_j) = 0. \tag{37}$$

Similarly, $u_j^{0,+}$ denotes the root of

$$\mu w_j x - t_j^0 - \rho f_j'(x - \hat{x}_j^0) = 0. \tag{38}$$

Thus we have $x_j^+ = \Pi_{[0,1]}(u_j^+)$ and $x_j^{0,+} = \Pi_{[0,1]}(u_j^{0,+})$. Now we introduce the notations $z_i^{(j)}$ and $\lambda_i^{(j)}$ to denote the j-th coordinate of $P_i^T \mathbf{z}_i$ and $P_i^T \boldsymbol{\lambda}_i$, respectively, then

$$t_j = -\gamma_j - \sum\nolimits_{i \in N(j)} (\lambda_i^{(j)} - \mu z_i^{(j)}),$$

where $N(j)$ denotes the set of all indices i that satisfy $H_{i,j} = 1$. From the assumption, $z_i^{0,(j)} = 1 - z_i^{(j)}$, $\lambda_i^{0,(j)} = -\lambda_i^{(j)}$, $\hat{x}_j^0 = 1 - \hat{x}_j$ and $\gamma_j^0 = -\gamma_j$ hold due to the symmetry of the channel. Therefore,

$$\begin{aligned} t_j^0 &= -\gamma_j^0 - \sum\nolimits_{i \in N(j)} (\lambda_i^{0,(j)} - \mu z_i^{0,(j)}) \\ &= \gamma_j - \sum\nolimits_{i \in N(j)} (-\lambda_i^{(j)} - \mu(1 - z_i^{(j)})) \\ &= \gamma_j + \sum\nolimits_{i \in N(j)} (\lambda_i^{(j)} - \mu z_i^{(j)}) + \mu \sum\nolimits_{i \in N(j)} 1 \\ &= -t_j + \mu w_j. \end{aligned}$$

We now verify $1 - u_j^+$ is the root of equation (38). Substituting $x = 1 - u_j^+$ in the left-hand side of equation (38), we have

$$\begin{aligned} &\mu w_j (1 - u_j^+) - t_j^0 - \rho f_j'(1 - u_j^+ - \hat{x}_j^0) \\ &\quad = -\mu w_j u_j^+ + t_j - \rho f_j'(\hat{x}_j - u_j^+) \\ &\quad = -\mu w_j u_j^+ + t_j - \rho f_j'(u_j^+ - \hat{x}_j) = 0, \end{aligned}$$

where the last equation is because u_j^+ is a root of (37) and f_j is symmetric about 0. We have shown that $u_j^+ = 1 - u_j^{0,+}$, it is clear that $\Pi_{[0,1]}(u_j^+) = \Pi_{[0,1]}(1 - u_j^{0,+})$. Thus we have $x_j^+ = x_j^{0,+}$.

Since the update rule for $\mathbf{z}$ and $\boldsymbol{\lambda}$ in Algorithm 1 is similar to that in [18], we refer the reader to [18, Lemma 18] for the rest of the proof. ∎

Lemma 5: We now use Algorithm 2 to decode two channel outputs, $\mathbf{y}$ and $\mathbf{y}^0$. Suppose that $\mathbf{x}^{[N]}$ denotes the N-th output of the ADMM algorithm when decoding $\mathbf{y}$, i.e., the output of the N-th iteration of Algorithm 2. Similarly, we denote the N-th output of the ADMM algorithm when decoding $\mathbf{y}^0$ as $\mathbf{x}^{0,[N]}$. If we initialize Algorithm 2 by letting $\mathbf{x}^0 = \mathbf{0.5}$, $\boldsymbol{\lambda}_i = \mathbf{0}$ for all i, $\hat{\mathbf{x}}^{[0]} = \mathbf{0.5}$, and set the convergence criterion of Algorithm 1 as round$(\mathbf{x}^k)$ = round$(\mathbf{x}^{k+1})$, then we have $\mathbf{x}^{[N]} = R_{\mathbf{c}}(\mathbf{x}^{0,[N]})$ for any N.

Proof: We use induction to prove our conclusion. Note that $\mathbf{x}$, $\boldsymbol{\lambda}_i$ and $\hat{\mathbf{x}}$ are initialized such that $\mathbf{x} = R_{\mathbf{c}}(\mathbf{x}^0) = \mathbf{0.5}$, $\mathbf{z}_i = R_{\mathbf{c}}(\mathbf{z}_i^0) = \mathbf{0.5}$, $\boldsymbol{\lambda}_i = -\boldsymbol{\lambda}_i^0 = \mathbf{0}$ and $\hat{\mathbf{x}} = R_{\mathbf{c}}(\hat{\mathbf{x}}^0) = \mathbf{0.5}$. By Lemma 4, we conclude that for every ADMM iteration k, we have

$$\mathbf{x}^{[1],k} = R_{\mathbf{c}}(\mathbf{x}^{0,[1],k}),$$

where $\mathbf{x}^{0,[1],k}$ denotes the variable $\mathbf{x}$ in the k-th iteration of Algorithm 1 with the input ρ and $\hat{\mathbf{x}}^{[0]}$. According to the assumption, the ADMM iteration stops when round$(\mathbf{x}^k)$ = round$(\mathbf{x}^{k+1})$. Therefore, $\mathbf{x}^{[1]} = \mathbf{x}^{[1],k_1+1}$ and $\mathbf{x}^{0,[1]} = \mathbf{x}^{0,[1],k_2+1}$, where k_1, k_2 are the smallest integers such that round$(\mathbf{x}^{[1],k_1})$ = round$(\mathbf{x}^{[1],k_1+1})$ and round$(\mathbf{x}^{0,[1],k_2})$ = round$(\mathbf{x}^{0,[1],k_2+1})$, respectively.

It remains to prove that round$(\mathbf{x}^{[1],k})$ = round$(\mathbf{x}^{[1],k+1})$ if and only if round$(\mathbf{x}^{0,[1],k})$ = round$(\mathbf{x}^{0,[1],k+1})$. It is true because for any real numbers a, b we have round(a) = round(b) if and only if round$(1-a)$ = round$(1-b)$.

We now assume that the conclusion holds for $1, 2, \ldots, N-1$. From the above proof, we only need to show that $\hat{\mathbf{x}}^{[N]} = R_{\mathbf{c}}(\hat{\mathbf{x}}^{0,[N]})$. By definition, we have

$$\begin{aligned} \hat{\mathbf{x}}^{[N]} &= \frac{\alpha}{\alpha + \beta^{[N]}} \boldsymbol{\zeta} + \frac{\beta}{\alpha + \beta^{[N]}} \frac{1}{S} \sum\nolimits_{l=L}^{N-1} \mathbf{x}^{[l]}, \\ \hat{\mathbf{x}}^{0,[N]} &= \frac{\alpha}{\alpha + \beta^{[N]}} \boldsymbol{\zeta} + \frac{\beta}{\alpha + \beta^{[N]}} \frac{1}{S} \sum\nolimits_{l=L}^{N-1} \mathbf{x}^{0,[l]}. \end{aligned}$$

From the assumptions we know that $\mathbf{x}^{[l]} = R_{\mathbf{c}}(\mathbf{x}^{0,[l]})$ for $l = 1, 2, \ldots, N-1$ and $\boldsymbol{\zeta} = R_{\mathbf{c}}(\boldsymbol{\zeta})$ by the definition of $\boldsymbol{\zeta}$. Thus we obtain the conclusion from the following lemma since $\hat{\mathbf{x}}^{[N]}(\hat{\mathbf{x}}^{0,[N]})$ is a convex combination of $\boldsymbol{\zeta}$ and $\mathbf{x}^{[l]}(\mathbf{x}^{0,[l]})$. ∎

Lemma 6: Let $\mathbf{x} = \sum_l \lambda_l \mathbf{x}^l$ and $\mathbf{y} = \sum_l \lambda_l \mathbf{y}^l$, where $\sum_l \lambda_l = 1$ and $\mathbf{x}^l = R_{\mathbf{c}}(\mathbf{y}^l)$, then $\mathbf{x} = R_{\mathbf{c}}(\mathbf{y})$.

Proof: We only need to consider indices j such that $c_j = 1$. Then $\mathbf{y}_j^l = 1 - \mathbf{x}_j^l$, and we have $y_j = \sum_l \lambda_l y_j^l = \sum_l \lambda_l (1 - x_j^l) = 1 - x_j$. ∎

The next lemma claims that there exists a one-to-one mapping between the output of our proposed algorithm for decoding any codeword $\mathbf{c}$ and the all-zero codeword.

Lemma 7: Let $\hat{\mathbf{c}}$ and $\hat{\mathbf{c}}^0$ be the output of Algorithm 2 for the channel output $\mathbf{y}$ and $\mathbf{y}^0$, respectively. Suppose that we apply the same initialization and convergence criterion as that stated in Lemma 5. Then $\hat{\mathbf{c}} = R_{\mathbf{c}}(\hat{\mathbf{c}}^0)$.

Proof: By Lemma 5, $\mathbf{x}^{[N]} = R_{\mathbf{c}}(\mathbf{x}^{0,[N]})$ for all N. It remains to prove that Algorithm 2 stops for decoding $\mathbf{y}$ and $\mathbf{y}^0$ at the same iteration. We denote $\hat{\mathbf{c}}^{[N]} := \text{round}(\mathbf{x}^{[N]})$ and $\hat{\mathbf{c}}^{0,[N]} := \text{round}(\mathbf{x}^{0,[N]})$. Note that Algorithm 2 stops whenever $[\mathbf{H}\hat{\mathbf{c}}^{[N]}]_2 = 0$. Thus we only have to verify $[\mathbf{H}\hat{\mathbf{c}}^{[N]}]_2 = 0$ if and only if $[\mathbf{H}\hat{\mathbf{c}}^{0,[N]}]_2 = 0$.

Since $\mathbf{x}^{[N]} = R_{\mathbf{c}}(\mathbf{x}^{0,[N]})$, we have $\hat{\mathbf{c}}^{[N]} = R_{\mathbf{c}}(\hat{\mathbf{c}}^{0,[N]})$. We now assume that $[\mathbf{H}\hat{\mathbf{c}}^{[N]}]_2 = 0$ to prove the "only if" part. To simplify the notation, we denote $\hat{\mathbf{c}}^{[N]}$ and $\hat{\mathbf{c}}^{0,[N]}$ by $\mathbf{r}$ and $\mathbf{r}^0$, respectively. Note that any row of $\mathbf{H}$ induces the equation $[r_{j_1} + r_{j_2} + \ldots + r_{j_d}]_2 = 0$, which means that there are even number of 1s among $r_{j_1}, \ldots, r_{j_d}$. Since

$$r_j^0 = \begin{cases} r_j, & \text{if } c_j = 0, \\ 1 - r_j, & \text{if } c_j = 1. \end{cases}$$

Therefore,

$$\begin{aligned} \left[\sum_k r_{j_k}^0\right]_2 &= \left[\sum_{\{k: c_{j_k} = 0\}} r_{j_k}\right]_2 + \left[\sum_{\{k: c_{j_k} = 1\}} (1 - r_{j_k})\right]_2 \\ &= \left[\sum_k r_{j_k}\right]_2 + \left[\sum_{\{k: c_{j_k} = 1\}} 1\right]_2 = 0. \end{aligned}$$

The last equation holds because $\mathbf{c}$ is a codeword and thus there are even numbers of 1s among $c_{j_1}, \ldots, c_{j_d}$. Therefore we finish the "only if" part of the proof. The "if" part is similar and we will not repeat it here. ∎

ACKNOWLEDGMENT

The authors thank Yixuan Hua and Haoming Liu for their helpful discussion and comments on the restartable ADMM.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[2] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, 1997.

[3] M. C. Davey and D. J. C. MacKay, "Low density parity check codes over GF(q)," in *Proc. Inf. Theory Workshop*, 1998, pp. 70–71.

[4] J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Commun.*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[5] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[6] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.

[7] K. Yang, X. Wang, and J. Feldman, "A new linear programming approach to decoding linear block codes," *IEEE Trans. Inf. Theory*, vol. 54, no. 3, pp. 1061–1072, Mar. 2008.

[8] M. Taghavi and P. Siegel, "Adaptive linear programming decoding," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2006, pp. 1374–1378.

[9] M. H. 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.

[10] S. Barman, X. Liu, S. C. Draper, and B. Recht, "Decomposition methods for large scale LP decoding," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 7870–7886, Dec. 2013.

[11] X. Jiao, J. Mu, Y.-C. He, and C. Chen, "Efficient ADMM decoding of LDPC codes using lookup tables," *IEEE Trans. Commun.*, vol. 65, no. 4, pp. 1425–1437, Apr. 2017.

[12] H. Wei, X. Jiao, and J. Mu, "Reduced-complexity linear programming decoding based on ADMM for LDPC codes," *IEEE Commun. Lett.*, vol. 19, no. 6, pp. 909–912, Jun. 2015.

[13] X. Jiao, Y. He, and J. Mu, "Memory-reduced look-up tables for efficient ADMM decoding of LDPC codes," *IEEE Signal Process. Lett.*, vol. 25, no. 1, pp. 110–114, Jan. 2018.

[14] F. Gensheimer, T. Dietz, S. Ruzika, K. Kraft, and N. Wehn, "A low-complexity projection algorithm for ADMM-based LP decoding," in *Proc. IEEE 10th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Dec. 2018, pp. 1–5.

[15] H. Wei and A. H. Banihashemi, "An iterative check polytope projection algorithm for ADMM-based LP decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 29–32, Jan. 2018.

[16] Q. Xia, Y. Lin, S. Tang, and Q. Zhang, "A fast approximate check polytope projection algorithm for ADMM decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 23, no. 9, pp. 1520–1523, Sep. 2019.

[17] J. Bai, Y. Wang, and Q. Shi, "Efficient QP-ADMM decoder for binary LDPC codes and its performance analysis," *IEEE Trans. Signal Process.*, vol. 68, pp. 503–518, 2020.

[18] X. Liu and S. C. Draper, "The ADMM penalized decoder for LDPC codes," *IEEE Trans. Inf. Theory*, vol. 62, no. 6, pp. 2966–2984, Jun. 2016.

[19] B. Wang, J. Mu, X. Jiao, and Z. Wang, "Improved penalty functions of ADMM penalized decoder for LDPC codes," *IEEE Commun. Lett.*, vol. 21, no. 2, pp. 234–237, Feb. 2017.

[20] H. Wei and A. H. Banihashemi, "ADMM check node penalized decoders for LDPC codes," *IEEE Trans. Commun.*, vol. 69, no. 6, pp. 3528–3540, Jun. 2021.

[21] Q. Wu, F. Zhang, H. Wang, J. Lin, and Y. Liu, "Parameter-free ℓ_p-box decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 22, no. 7, pp. 1318–1321, Jul. 2018.

[22] Y. Wei, M.-M. Zhao, M.-J. Zhao, and M. Lei, "A PDD decoder for binary linear codes with neural check polytope projection," *IEEE Wireless Commun. Lett.*, vol. 9, no. 10, pp. 1715–1719, Oct. 2020.

[23] B. Jiang, Y.-F. Liu, and Z. Wen, "L_p-norm regularization algorithms for optimization over permutation matrices," *SIAM J. Optim.*, vol. 26, no. 4, pp. 2284–2313, 2016.

[24] Y. Wang, W. Yin, and J. Zeng, "Global convergence of ADMM in nonconvex nonsmooth optimization," *J. Scientific Comput.*, vol. 78, no. 1, pp. 29–63, Jan. 2019.

[25] H. Attouch, J. Bolte, and B. F. Svaiter, "Convergence of descent methods for semi-algebraic and tame problems: Proximal algorithms, forward–backward splitting, and regularized Gauss–Seidel methods," *Math. Program.*, vol. 137, nos. 1–2, pp. 91–129, Feb. 2013.

Yiming Chen received the B.Sc. degree in mathematics from Sichuan University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree in computational mathematics with Peking University. His research interests include combinatorial optimization, stochastic optimization, and their applications in machine learning.

Rui Wang received the B.Sc. degree in information and computing science and the Ph.D. degree in operational research from Beijing Jiaotong University, Beijing, China, in 2015 and 2021, respectively. She is currently a Post-Doctoral Researcher with Peking University, Beijing. Her current research interests include large-scale classification optimization problems, sparse optimization, and numerical computing.

Jinglong Zhu received the B.Sc. and Ph.D. degrees in mathematics from Peking University in 2012 and 2017, respectively. He is currently a PHY Algorithm Design Engineer with Huawei HiSilicon. His main research interests include application of novel mathematical methods in low complexity algorithm design.

Zaiwen Wen received the Ph.D. degree in operations research from Columbia University in 2009. He is currently a Professor with Peking University. His research interests include optimization algorithms and theory and their applications in machine learning. He received the China Youth Science and Technology Award in 2016 and the Beijing Outstanding Youth Zhongguancun Award in 2020. He was funded by the National Ten Thousand Talents Program for Science. He is an Associate Editor of *Journal of Scientific Computing*, *Communications in Mathematics and Statistics*, *Journal of the Operations Research Society of China*, and *Journal of Computational Mathematics*, and a Technical Editor of *Mathematical Programming Computation*.