ADMM and Reproducing Sum-Product Decoding Algorithm Applied to QC-MDPC Code-based McEliece Cryptosystems

Kohtaro Watanabe†, Motonari Ohtsuka and Yuta Tsukie
Department of Computer Science, National Defense Academy of Japan
† wata@nda.ac.jp

***Abstract*—QC-MDPC (quasi cyclic moderate density parity check) code-based McEliece cryptosystems are considered to be one of the candidates for post-quantum cryptography. Decreasing DER (decoding error rate) is one of important factor for their security, since recent attacks to these cryptosystems effectively use DER information. In this paper, we pursue the possibility of optimization-base decoding, concretely we examine ADMM (alternating direction method of multipliers), a recent developing method in optimization theory. Further, RSPA (reproducing sum-product algorithm), which efficiently reuse outputs of SPA (sum-product algorithm) is proposed for the reduction of execution time in decoding. By numerical simulations, we show that the proposing scheme shows considerable decrement in DER compared to the conventional decoding methods such as BF (bit-flipping algorithm) or SPA.**

***Index Terms*—QC-MDPC code-based cryptosystem, ADMM method, reproducing sum-product algorithm, McEliece cryptosystem.**

I. Introduction

The safety of current cryptosystem (for example, RSA, ECDH) are based on the computation difficulty such as integer factoring or discrete logarithmic problem. These crypto schemes are said to be no longer secure under the presence of quantum computers, which are able to solve these problems in polynomial time [16]. Although current number of qubits are relatively small, algorithms resistant for quantum computers equipped with practical qubits are desirable. Following proposals; lattice-based, code-based, hash-based and multivariate, isogeny-base etc. are considered to be the framework of cryptosystems for post quantum generation and these are continuing to be reviewed in NIST post-quantum cryptography standardization [20].

The present paper is concerned with the reduction of DER (decoding error rate) in code-based cryptosystem. It is known that there is an attack (GJS attack) [8] that effectively uses decoding failure statistics to retrieve the parity check matrix of the code (which is a secret key). Thus, reducing the DER is considered to be one of important factors for preventing such attacks. We note the efforts to reduce DER are seen in recent studies [5], [6], [10], [13]–[15].

The security of code-based schemes is based on the hardness of decoding linear random codes (indeed the problem is known to be NP-hard problem [3]). The first code-based scheme is the McEliece cryptosystem which use Goppa codes. The original McEliece cryptosystem has been withstanding evaluation more than thirty years and still regarded secure. A problem of original McEliece cryptosystem is its large public key size. For the reduction of public key size, various improvements were proposed; see [4], [12], [19]. It should be noted that the usage of LDPC codes directly for this purpose is known to be dangerous, since parity check matrix of LDPC (low-density parity check) codes might be recovered by decoding low Hamming weight codewords, for example using Stern's method [17], as dual codes of original codes.

On the other hand, QC-MDPC (quasi cyclic moderate density parity check) code is currently regarded as safe and public key size is small, and for such reasons, this scheme is remaining as alternatives of NIST PQC as Public-key Encryption and Key-establishment Algorithms [20]. Standard decoding methods for QC-MDPC codes are variants of BF (bit-flipping) algorithm. In this paper, we show DER of QC-MDPC codes can be considerably improved compared to BF or SPA (sum-product algorithm) by applying ADMM (alternating direction method of multipliers) based decoding, which is a recently developing subject in optimization theory. The ADMM decoding is a method based on IP (integer programming), which achieve MLD (maximum likelihood decoding); see Feldman, Wainwright and Karger [7]. Although IP decoding presents high precision decoding, it costs a considerable amount of execution time and hence the adoption seems to be limited to relatively short code-length problems (about a few hundred bits or so). While the ADMM decoding is reported to be applicable to more than thousands of bits of code-length; see [2], [11]. In this paper, we apply the algorithm of Barman, Liu, Draper and Recht [2] and Liu and Draper [11] as the ADMM decoding method and show even in the case of middle-density parity checks and middle code-length (in cases code-length are 9602 and 10779), it works well.

Here, we have to note that the original projection algorithm (Algorithm 2 of [2]) seems to be including some error (actually, Algorithm 2 of [2] sometimes shows mismatching values compared to the values computed by MIP solver Gurobi optimizer 9.5 [9]); see Figure 1 and 2 in Section III. We present the corrected projection algorithm to the code-polytope in Algorithm 3 and examine its validity in Section VI.

The ADMM method can decode encrypted messages considerably faster than original IP decoding, however, in our

implementation, it required about 2.8∼7.8 times as many decoding time in average, compared to BF algorithm; see Table I and II. For improving this difference, we propose yet another decoding method, RSPA (reproducing sum-product algorithm), which efficiently reuse decoding failure of SPA. As a result, the combination of RSPA and ADMM decoding show the certain improvement of decoding time in average.

As for the reduction in DER, by numerical examination in Section IV, we observe in the case of $(n, k) = (9602, 4801)$ code, the correctable number of errors increases from $t = 87$ (BF algorithm) to $t = 103$ under almost the same DER (less than 10^{-7}). Similarly, in the case of $(n, k) = (10779, 3593)$, correctable number of number of errors increases from $t = 55$ (BF algorithm) to $t = 66$ under almost the same DER (less than 10^{-7}). In other words, proposing scheme attains far less DER for fixed error number t compared to BF algorithm and hence prevents the collection of DER information by adversaries.

II. QC-MDPC McEliece cryptosystems

A. McEliece cryptosystems

We briefly review McEliece cryptosystems. Throughout this paper, a set of codewords is denoted by $\mathcal{C}$. The variables n, k, t denote code-length, information bits length and correctable error number, respectively. We abbreviate $m = n - k$. Also, $k \times n$ binary matrix $\boldsymbol{G}$, $m \times n$ binary matrix $\boldsymbol{H}$, $n \times n$ matrix $\boldsymbol{P}$ and $k \times k$ binary matrix $\boldsymbol{S}$ represent generator matrix, parity-check matrix, random permutation matrix and scramble matrix respectively. Put

$$\boldsymbol{G'} = \boldsymbol{SGP}. \tag{1}$$

Then $(\boldsymbol{G}', t)$ is a public key and $(\boldsymbol{G}, \boldsymbol{S}, \boldsymbol{P})$ is a private key of Bob. For a message $\boldsymbol{m}$, using public key, Alice encrypts as

$$\boldsymbol{c} = \boldsymbol{mG'} + \boldsymbol{e}, \tag{2}$$

where $\boldsymbol{e}$ is a correctable error vector whose Hamming weight satisfies $w(\boldsymbol{e}) = t$. Decoding process of Bob is as follows:

(i) Multiply $\boldsymbol{P}^{-1}$ to both sides of (2), that is

$$\boldsymbol{cP^{-1}} = \boldsymbol{mG'P^{-1}} + \boldsymbol{eP^{-1}} = \boldsymbol{mSG} + \boldsymbol{eP^{-1}}.$$

(ii) Since $w(\boldsymbol{eP^{-1}}) = w(\boldsymbol{e}) = t$, by applying certain decoding scheme, he obtains $\boldsymbol{c'} = \boldsymbol{mSG}$.

(iii) Multiplying $\boldsymbol{G^{-1}S^{-1}}$ to $\boldsymbol{c'}$, Bob retrieves the sent message $\boldsymbol{m}$.

B. QC-MDPC McEliece cryptosystems with code-rate $R = (n_0 - 1)/n_0$

QC-MDPC codes are linear block codes whose binary parity check matrices have moderate density of "one' and are introduced to reduce public key size of original McEliece cryptosystems; see Misoczki, Tillich, Sendier and Barreto [12]. A parity check matrix of QC-MDPC McEliece cryptosystems with code-rate $R = (n_0 - 1)/n_0$ assumes the following form; see for example [1, Section 3.6],

$$\boldsymbol{H} = [\boldsymbol{H_0}, \boldsymbol{H_1}, \ldots, \boldsymbol{H_{n_0-1}}], \tag{3}$$

where each $\boldsymbol{H_i}$, $(0 \le i \le n_0 - 1)$ is a binary $p \times p$ circulant matrix of each row and column weight equals to d_c. Hence it holds that $n = n_0 p$ and $k = (n_0 - 1)p$ (so $R = (n_0 - 1)/n_0$). Without loss of generality, we can assume $\boldsymbol{H}_{n_0-1}$ is non-singular, then the generator matrix of the code is expressed as (see also [1, Section 3.6]),

$$\boldsymbol{G} = \left[\begin{array}{c|c} \boldsymbol{I_k} & \begin{array}{c} \left(\boldsymbol{H_{n_0-1}^{-1}} \cdot \boldsymbol{H_0}\right)^T \\ \vdots \\ \left(\boldsymbol{H_{n_0-1}^{-1}} \cdot \boldsymbol{H_{n_0-2}}\right)^T \end{array} \end{array}\right], \tag{4}$$

where $\boldsymbol{I_k}$ is the $k \times k$ identity matrix. Since $\left(\boldsymbol{H_{n_0-1}^{-1}} \cdot \boldsymbol{H_i}\right)^T$, $(0 \le i \le n_0 - 1)$ are circulant, public key will be only first rows of these matrices, and as a result, reducing the size of the public key.

III. Decoding Methods

A. Bit-flipping decoding algorithm

The basic bit-flipping decoding algorithm is described as follows:

Algorithm 1 Bit-flipping decoding algorithm

```
s ← Hc^T.
while s ≠ 0 do
    i ← arg min_{0≤i≤n−1} w(s + h_i).
    if w(s + h_i) < w(s) then
        c_i ← 1 − c_i.
        s ← s + h_i.
    else
        return decoding failure.
    end if
end while
return c.
```

As shown in the numerical experiments of next section, BF decoding algorithm shows lower DER than SPA, especially in cases $w(\boldsymbol{e})$ are relatively small value; see Figure 7 and 8.

B. Notations

Next, we introduce ADMM algorithm according to [2]. Although, an algorithm developed in [2] assumes LDPC codes, we show that the algorithm is also effective for decoding of MDPC codes. We define some necessary notations: $\mathcal{X}$ and $\mathcal{Y}$ denote: $\mathcal{X} = \{0, 1\}$ and $\mathcal{Y} = \{0, 1\}$, since we assume BSC (binary symmetric channel). Thus, sending messages belong to k-dimensional subspace $\mathcal{C}$ of $\mathcal{X}^n$ and corresponding received messages belong to $\mathcal{Y}^n$. The neighborhood of a each "check" i is denoted by $N_c(i)$ and the neighborhood of a each "vertex" j is denoted by $N_v(j)$, namely:

$$N_c(i) = \{j \,|\, H_{i,j} = 1\} \text{ and } N_v(j) = \{i \,|\, H_{i,j} = 1\},$$

where $H_{i,j}$ is a (i, j) element of parity check matrix $\boldsymbol{H}$. In the case, H is assumed to be as (3), since each row and column of $\boldsymbol{H}_k$, $(0 \le k \le n_0 - 1)$ has d_c non-zero element, it holds

that $|N_c(i)| = n_0 d_c$, $(0 \le i \le p)$ and $|N_v(j)| = d_c$, $(0 \le j \le n_0 p)$.

Let $N_c(i) = \{j_1, j_2, \cdots, j_d\}$ (note that in the case of (3), $d = n_0 d_c$), then we define a linear map $\boldsymbol{P}_i : \mathbb{R}^n \to \mathbb{R}^d$ as

$$(\boldsymbol{P}_i)_{k,l} = \begin{cases} 1, & l = j_k \\ 0, & \text{else.} \end{cases}$$

For example, in the case

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{5}$$

we have

$$\boldsymbol{P}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \ \boldsymbol{P}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\boldsymbol{P}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Define

$$\mathbb{P}_d := \left\{ \boldsymbol{x} \in \{0,1\}^d \,|\, \|\boldsymbol{x}\|_1 \text{ is even} \right\},$$

where $\|\cdot\|_p$ represents a l^p norm. Then, we can write

$$\boldsymbol{x} \in \mathcal{C} \Leftrightarrow \boldsymbol{P}_i \boldsymbol{x} \in \mathbb{P}_d, \ (1 \le i \le m).$$

For $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, $W(y|x)$ represents the conditional probability of BSC, i.e. $W(1|0) = W(0|1) = p$ and $W(0|0) = W(1|1) = 1 - p$ where $p \in [0, 1/2)$. Assume a sending message is $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathcal{X}^n$ and a received message be $\boldsymbol{y} = (y_1, \ldots, y_n) \in \mathcal{Y}^n$. Since we assume memoryless channel, the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$ is represented as: $p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{j=1}^n W(y_j|x_j)$. The ML encoding choose $\boldsymbol{x} \in \mathcal{X}^n$ that maximizes $p(\boldsymbol{y}|\boldsymbol{x})$, thus equivalently maximizes $\sum_{j=1}^n \log W(y_j|x_j)$. We put

$$\gamma_j := \log \left(W(y_j|0)/W(y_j|1) \right), \quad (1 \le j \le n). \tag{6}$$

Since $\log W(y_j|x_j) = -\gamma_j x_j + \log W(y_j|0)$, ML decoding reduces to determining $\boldsymbol{x} \in \mathcal{C}$ that minimizes $\boldsymbol{\gamma}^T \boldsymbol{x} = \sum_{j=1}^n \gamma_j x_j$. Thus, ML decoding is described as:

Minimize:
$$\sum_{j=1}^n \gamma_j x_j \ \text{ subject to } \boldsymbol{P}_i \boldsymbol{x} \in \mathbb{P}_d, \ (1 \le i \le m). \tag{7}$$

By the relaxation of each binary variable x_i $(1 \le i \le n)$ to the interval $[0, 1]$, we obtain LP (linear programming) relaxation of IP (7) (The LP relaxation by Feldman etc. [7] is a kind of this relaxation):

Minimize:
$$\sum_{j=1}^n \gamma_j x_j \ \text{ subject to } \boldsymbol{P}_i \boldsymbol{x} \in \mathbb{PP}_d, \ (1 \le i \le m), \tag{8}$$

where $\mathbb{PP}_d$ is a convex hull of $\mathbb{P}_d$, i.e. $\mathbb{PP}_d = \text{conv}(\mathbb{P}_d)$.

C. ADMM formulation

For expressing LP decoding formulation (8) to corresponding ADMM formulation, the "replica" vectors $\boldsymbol{z}_i$, $(1 \le i \le m)$ are introduced. Moreover, to accelerate the computation, we also introduce a penalty term. Hence, we consider

Minimize:
$$\sum_{j=1}^n \gamma_j x_j - \alpha \sum_{j=1}^n (x_j - 0.5)^2 \tag{9}$$
subject to $\boldsymbol{P}_i \boldsymbol{x} = \boldsymbol{z}_i, \ \boldsymbol{z}_i \in \mathbb{PP}_d, \ (1 \le i \le m)$,

where $\alpha > 0$ is an acceleration parameter and the second term of objective function is a penalty term. Put $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m)$ and $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_m)$. The ADMM method optimize the augmented Lagrangian:

$$\begin{aligned} L_\mu(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) := & \sum_{j=1}^n \gamma_j x_j - \alpha \sum_{j=1}^n (x_j - 0.5)^2 \\ & + \sum_{i=1}^m \boldsymbol{\lambda}_i^T (\boldsymbol{P}_i \boldsymbol{x} - \boldsymbol{z}_i) + \frac{\mu}{2} \sum_{i=1}^m \|\boldsymbol{P}_i \boldsymbol{x} - \boldsymbol{z}_i\|_2^2, \end{aligned} \tag{10}$$

We can write the update steps of ADMM as:

$$\begin{aligned} \boldsymbol{x}^{q+1} &:= \arg \min_{\boldsymbol{x} \in [0,1]^n} L_\mu (\boldsymbol{x}, \boldsymbol{z}^q, \boldsymbol{\lambda}^q) \\ \boldsymbol{z}_i^{q+1} &:= \arg \min_{\boldsymbol{z} \in (\mathbb{PP}_d)^m} L_\mu \left(\boldsymbol{x}^{q+1}, \boldsymbol{z}, \boldsymbol{\lambda}^q \right), \ (1 \le i \le m) \\ \boldsymbol{\lambda}_i^{q+1} &:= \boldsymbol{\lambda}_i^q + \mu \left(\boldsymbol{P}_i \boldsymbol{x}^{q+1} - \boldsymbol{z}_i^{q+1} \right), \ (1 \le i \le m). \end{aligned} \tag{11}$$

Let $\boldsymbol{\delta x} \in \mathbb{R}^n$ be a variation vector of $\boldsymbol{x}$, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n)$ and $\mathbf{1} = (1, \ldots, 1)^T$. Then, we obtain

$$\begin{aligned} & L_\mu (\boldsymbol{x} + \boldsymbol{\delta x}, \boldsymbol{z}, \boldsymbol{\lambda}) = L_\mu(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) + \langle \boldsymbol{\gamma}, \boldsymbol{\delta x} \rangle \\ & - 2\alpha \langle \boldsymbol{x} - 0.5 \cdot \mathbf{1}, \boldsymbol{\delta x} \rangle + \sum_{i=1}^m \langle \boldsymbol{\lambda}_i, \boldsymbol{P}_i \boldsymbol{\delta x} \rangle \\ & + \mu \sum_{i=1}^m \langle \boldsymbol{P}_i \boldsymbol{x} - \boldsymbol{z}_i, \boldsymbol{P}_i \boldsymbol{\delta x} \rangle \\ & = \langle \boldsymbol{\gamma}, \boldsymbol{\delta x} \rangle - 2\alpha \langle \boldsymbol{x} - 0.5 \cdot \mathbf{1}, \boldsymbol{\delta x} \rangle + \sum_{i=1}^m \langle \boldsymbol{P}_i^T \boldsymbol{\lambda}_i, \boldsymbol{\delta x} \rangle \\ & + \mu \sum_{i=1}^m \langle \boldsymbol{P}_i^T \boldsymbol{P}_i \boldsymbol{x} - \boldsymbol{P}_i^T \boldsymbol{z}_i, \boldsymbol{\delta x} \rangle \\ & = \langle \boldsymbol{\gamma}, \boldsymbol{\delta x} \rangle - 2\alpha \langle \boldsymbol{x} - 0.5 \cdot \mathbf{1}, \boldsymbol{\delta x} \rangle + \sum_{i=1}^m \langle \boldsymbol{P}_i^T \boldsymbol{\lambda}_i, \boldsymbol{\delta x} \rangle \\ & + \langle \mu \boldsymbol{P} \boldsymbol{x} - \mu \sum_{i=1}^m \boldsymbol{P}_i^T \boldsymbol{z}_i, \boldsymbol{\delta x} \rangle, \end{aligned}$$

where $\boldsymbol{P} = \sum_{i=1}^m \boldsymbol{P}_i^T \boldsymbol{P}_i$. Thus, we obtain,

$$\left(\boldsymbol{P} - \frac{2\alpha}{\mu} \boldsymbol{I} \right) \boldsymbol{x} = \sum_{i=1}^m \left(\boldsymbol{P}_i^T \boldsymbol{z}_i - \frac{1}{\mu} \boldsymbol{P}_i^T \boldsymbol{\lambda}_i \right) - \frac{\alpha}{\mu} \mathbf{1} - \frac{\boldsymbol{\gamma}}{\mu}. \tag{12}$$

In the case of (5), we have

$$\boldsymbol{P} = \sum_{i=1}^{3} \boldsymbol{P}_i^T \boldsymbol{P}_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix},$$

and in general, we have $(\boldsymbol{P})_{i,j} = |N_v(j)|\delta_{i,j}$. For simplicity, we denote by $z_i^{(j)}$, the j-th component of $\boldsymbol{P}_i^T \boldsymbol{z}_i$. Similarly, $\lambda_i^{(j)}$ denotes the j-th component of $\boldsymbol{P}_i^T \boldsymbol{\lambda}_i$. Hence from (12) and the restriction $x_j \in [0,1]$ for $(1 \leq j \leq n)$, it holds that

$$\begin{aligned} x_j =& \Pi_{[0,1]} \left(\frac{1}{|N_v(j)| - \frac{2\alpha}{\mu}} \right) \cdot \\ & \cdot \left\{ \sum_{i=1}^{m} \left(z_i^{(j)} - \frac{\lambda_i^{(j)}}{\mu} \right) - \frac{\alpha + \gamma_j}{\mu} \right\}, \quad (1 \leq j \leq n), \end{aligned} \tag{13}$$

where $\Pi_{[0,1]}$ is a projection to the interval $[0,1]$. To obtain the minimizer of the second expression of (11), we rewrite the third and fourth terms of (10) as follows:

$$\begin{aligned} & \boldsymbol{\lambda}_i^T (\boldsymbol{P}_i \boldsymbol{x} - \boldsymbol{z}_i) + \frac{\mu}{2} \|\boldsymbol{P}_i \boldsymbol{x} - \boldsymbol{z}_i\|_2^2 \\ &= \frac{\mu}{2} \left\{ \|\boldsymbol{P}_i \boldsymbol{x} + \frac{\boldsymbol{\lambda}_i}{\mu} - \boldsymbol{z}_i\|_2^2 - \frac{\|\boldsymbol{\lambda}_i\|_2^2}{\mu^2} \right\}. \end{aligned}$$

Hence it holds

$$\boldsymbol{z}_i^{q+1} = \arg \min_{\boldsymbol{z}_i \in \mathbb{PP}_d} \|\boldsymbol{P}_i \boldsymbol{x}^{q+1} + \frac{\boldsymbol{\lambda}_i^q}{\mu} - \boldsymbol{z}_i\|_2,$$

which is a metric projection of $\boldsymbol{P}_i \boldsymbol{x}^{q+1} + \frac{\boldsymbol{\lambda}_i^q}{\mu}$ onto the closed convex set $\mathbb{PP}_d$ with $d = N_0 d_c$, i.e.

$$\boldsymbol{z}_i^{q+1} = \Pi_{\mathbb{PP}_d} \left(\boldsymbol{P}_i \boldsymbol{x}^{q+1} + \frac{\boldsymbol{\lambda}_i^q}{\mu} \right). \tag{14}$$

Summarizing the above argument, the ADMM decoding procedure is described as follows. Recall that $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e}$ is a received message where $\boldsymbol{e}$ satisfies $w(\boldsymbol{e}) = t$.

Algorithm 2 The ADMM decoding algorithm

1: Initialize $\boldsymbol{\gamma}$ as (6), $q = 0$, set a positive integer M and $\epsilon > 0$, a small number.
2: Initialize $\boldsymbol{z}_i = \boldsymbol{P}_i \boldsymbol{y}$, $\boldsymbol{\lambda}_i = 0$, $(1 \leq i \leq m)$.
3: **repeat**
4: **for all** j $(1 \leq j \leq n)$ **do**
5: Update x_j accrding to (13).
6: **end for**
7: **for all** i $(1 \leq i \leq m)$ **do**
8: **set** $\boldsymbol{v}_i = \boldsymbol{P}_i \boldsymbol{x} + \boldsymbol{\lambda}_i / \mu$.
9: $\boldsymbol{z}_i \leftarrow \Pi_{\mathbb{PP}_{n_0 d_c}}(\boldsymbol{v}_i)$.
10: $\boldsymbol{\lambda}_i \leftarrow \boldsymbol{\lambda}_i + \mu (\boldsymbol{P}_i \boldsymbol{x} - \boldsymbol{z}_i)$.
11: **end for**
12: $q \leftarrow q + 1$
13: **until** $H\boldsymbol{x}^T = \boldsymbol{0}$ (Parity satisfied) **or** $q > M$
14: **return** $\boldsymbol{x}$.

The most time consuming part of the Algorithm 2 is Step 9: $\boldsymbol{z}_i \leftarrow \Pi_{\mathbb{PP}_{n_0 d_c}}(\boldsymbol{v}_i)$, a projection of $\boldsymbol{v}_i$ to code-polytope $\mathbb{PP}_{n_0 d_c}$.

Here, we have to note the projection algorithm to the code-polytope $\mathbb{PP}_{n_0 d_c}$ (Algorithm 2 of [2]) seems to need some modification, and without such modification, non-negligible degradation in decoding performance arises. Indeed, sometimes mismatching values were observed for the convex-quadratic programming (20) compared to the one obtained by MIP solver Gurobi optimizer 9.5 [9] (we note replacing the projection algorithm for (20) with MIP solver is unrealistic from the time-consuming view point).

We will show these degradations with numerical examples. Figure 1 shows the DER of each decoding method for $(n,k) = (2000, 1000)$ and $d_c = 5$ LDPC code. In the figure labels "SPA" and "BF" mean sum-product algorithm and bit-flipping algorithm, respectively. While labels "PROJ$_{\mathrm{O}}$" and "PROJ" mean ADMM decoding with original projection algorithm of [2] (Algorithm 2) and corrected projection algorithm respectively. We can observe that "PROJ" decreases DER about $1/10$ times to "PROJ$_{\mathrm{O}}$" for each error number t and even superior to SPA. We note both ADMM decoding use parameters $\alpha = 10$, $\mu = 6$ and $M = 10000$. Figure 2 shows the DER of each decoding method for $(n,k) = (2000, 1000)$ and $d_c = 25$ MDPC code. In this case SPA deteriorates its performance compared to the of Figure 1 since the low-density property of parity check matrix was lost. Comparing "PROJ$_{\mathrm{O}}$" with "PROJ", we observe that in this case "PROJ" also decreases DER about $1/10$ times to "PROJ$_{\mathrm{O}}$" for each error number t. We note both ADMM decoding use parameters $\alpha = 10$, $\mu = 6$ and $M = 10000$.

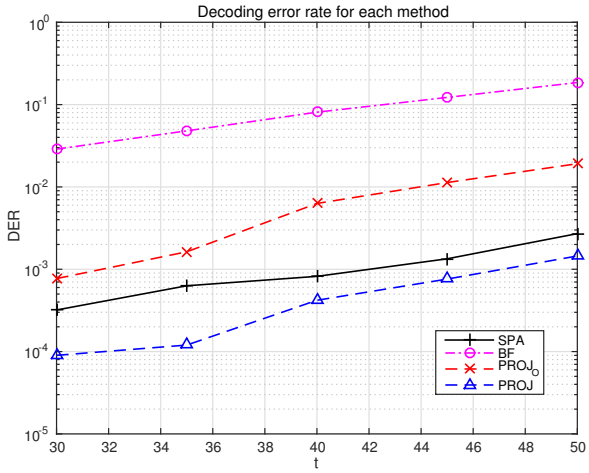

Fig. 1. DER of each decoding method for $(n,k) = (2000, 1000)$ and $d_c = 5$ LDPC code.

From these observations, we conclude that the correction of projection algorithm is a principal factor for the improvement of DER.

D. Modified projection algorithm

Following is a correction of projection algorithm to $\mathbb{PP}_{n_0 d_c}$. For simplicity, we abbreviate $d = N_0 d_c$ and denote by $\lfloor x \rfloor_{\text{even}}$ the largest even number integer smaller than x. The correctness of the following algorithm is shown in Appendix VI.

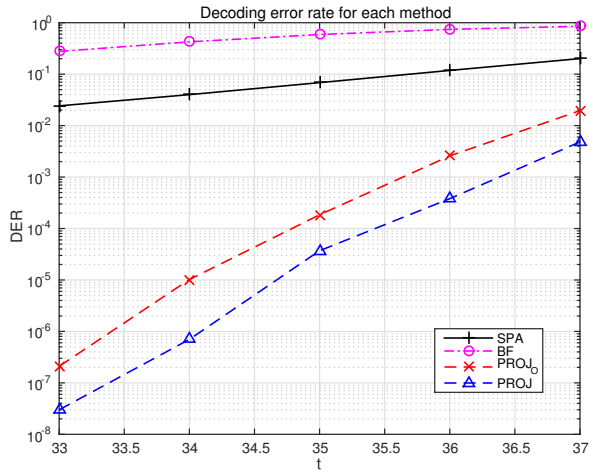

Fig. 2. DER of each decoding method for $(n,k)=(2000,1000)$ and $d_c=25$ MDPC code.

Algorithm 3 The projection algorithm $\Pi_{\mathbb{PP}_d}(\boldsymbol{u})$

Permutate $\boldsymbol{u}$ to $\boldsymbol{v}$ in descending order: $v_1 \geq v_2 \geq \cdots \geq v_d$. Let $\boldsymbol{Q}$ be the corresponding permutation matrix i.e. $\boldsymbol{v}=\boldsymbol{Q}\boldsymbol{u}$.
$\hat{\boldsymbol{z}} \leftarrow \Pi_{[0,1]^d}(\boldsymbol{v})$. Put $r=\lfloor \|\hat{\boldsymbol{z}}\|_1 \rfloor_{\text{even}}$.
Put $\boldsymbol{f}=(f_1,\ldots,f_d)$ and **set** $f_i=1$, $(1\leq i\leq r+1)$, $f_i=-1$, $(r+2\leq i\leq d)$.
if $r=d$ **or** $r=d-1$ **then**
  return $\boldsymbol{Q}^T\hat{\boldsymbol{z}}$.
end if
if $\sum_{i=1}^d f_i\hat{z}_i \leq r$ **then**
  return $\boldsymbol{Q}^T\hat{\boldsymbol{z}}$.
end if
set $\beta_{\max} \leftarrow \frac{1}{2}(v_{r+1}-v_{r+2})$.
set $\hat{\boldsymbol{\beta}} \leftarrow \left\{\hat{\beta}_1,\ldots,\hat{\beta}_{2d}\right\}$ as

$$\hat{\beta}_i=\begin{cases} v_i-1, & (1\leq i\leq r+1)\\ -v_i, & (r+2\leq i\leq d)\\ v_i, & (d+1\leq i\leq d+r+1)\\ 1-v_i, & (d+r+2\leq i\leq 2d).\end{cases}$$

set $\tilde{\boldsymbol{\beta}} \leftarrow \left\{\tilde{\beta}\in\hat{\boldsymbol{\beta}}\cup\{0\}\cup\{\beta_{\max}\}\,|\,0\leq\tilde{\beta}\leq\beta_{\max}\right\}$.
Sort $\tilde{\boldsymbol{\beta}}$ to $\boldsymbol{\beta}$ in ascending order. Hence $\boldsymbol{\beta}=\{\beta_1,\beta_2,\ldots\beta_p\}$ satisfies $0=\beta_0<\beta_1<\cdots<\beta_p=\beta_{\max}$.
set $L\leftarrow 0, R\leftarrow p, M\leftarrow\lfloor(L+R)/2\rfloor$.
while $R-L>1$ **do**
  set

$$\hat{z}_i=\begin{cases} v_i-\beta_M, & (1\leq i\leq r+1)\\ v_i+\beta_M, & (r+2\leq i\leq d).\end{cases}$$

  if $\sum_{i=1}^d f_i\cdot\left(\Pi_{[0,1]}(\hat{z}_i)\right)<r$ **then**
    $R\leftarrow M$.
  else
    $L\leftarrow M$.
  end if
  $M\leftarrow\lfloor(L+R)/2\rfloor$.
end while
set

$$\hat{z}_i=\begin{cases} v_i-\beta_R, & (1\leq i\leq r+1)\\ v_i+\beta_R, & (r+2\leq i\leq d).\end{cases}$$

set $f_z\leftarrow\sum_{i=1}^d f_i\cdot\left(\Pi_{[0,1]}(\hat{z}_i)\right)$.
set

$$\hat{z}_i=\begin{cases} v_i-\beta_L, & (1\leq i\leq r+1)\\ v_i+\beta_L, & (r+2\leq i\leq d).\end{cases}$$

set $f_{z_0}\leftarrow\sum_{i=1}^d f_i\cdot\left(\Pi_{[0,1]}(\hat{z}_i)\right)$.
set $\beta_{\text{opt}}\leftarrow\frac{(r-f_z)(\beta_R-\beta_L)}{f_z-f_{z_0}}+\beta_R$
set z_i, $(1\leq i\leq d)$ as

$$z_i=\begin{cases}\Pi_{[0,1]}(v_i-\beta_{\text{opt}}), & (1\leq i\leq r+1)\\ \Pi_{[0,1]}(v_i+\beta_{\text{opt}}), & (r+1\leq i\leq d).\end{cases}$$

return $\boldsymbol{Q}^T\boldsymbol{z}$.

IV. Numerical Experiments

We examine the proposing ADMM based decoding with some parameters of Table II of [12]. Numerical examinations were executed on Intel Core i9-7980XE CPU @ 2.60GHz processor, with no parallelization (except Step 8 in Algorithm 5 where N_{SPA} parallelization were applied) and ANSI C implementation (icc 2021.6.0 with -O3 option). Table I shows the execution time ratio (execution time of BF was taken as a reference point) for $(n,k,d_c,n_0)=(9802,4601,45,2)$ code. The number of experiments for each method is 100000. In SP, maximum iteration number is set as 50 and in ADMM, parameters are set as $(\mu,\alpha,M)=(6,15,10000)$. The label "ME" is the "matrix-extension" method proposed in [5]. The maximum extension size of parity matrix is 14403×19204. For $(n,k,d_c,n_0)=(10779,3593,51,3)$ code, execution time ratio is as Table II. In SP, maximum iteration number is set as 50 and in ADMM, parameters are set as $(\mu,\alpha,M)=$

TABLE I
Execution time ratio of each decoding methods for $(n,k,d_c,n_0)=(9802,4601,45,2)$ code.

method	$t=105$	$t=100$
BF	1.0	1.0
SPA	1.90	1.53
ADMM	3.60	2.76
ME	2.84	1.67
RSPA-ADMM	1.98	1.55

TABLE II
Execution time ratio of each decoding methods for $(n,k,d_c,n_0)=(10779,3593,51,3)$ code.

method	$t=68$	$t=66$
BF	1.0	1.0
SPA	2.53	2.04
ADMM	7.76	5.80
ME	1.99	1.15
RSPA-ADMM	4.02	2.19

$(8, 15, 10000)$. In this case the maximum extension size of parity matrix in ME is 14372×21588.

Although as we can recognize from Figures 2, 7 and 8, error correction ability of ADMM is superior to BF SP and ME, average decoding time, in our implementation, requires about 2.8∼7.8 as many times compared to BF decoding. For improving this difference, we pursue yet another algorithm RSPA (Reproducing Sum-Product Algorithm) which effectively reuse outputs of SPA. The algorithm RSPA is as follows:

Algorithm 4 The reproducing sum-product algorithm

1: **set** N_{SPA} as maximum iteration number, $i = 0$.
2: **while** $i < N_{\mathrm{SPA}}$ **do**
3: **set** $\boldsymbol{c}'$ as decoding result of SPA for $\boldsymbol{c}$ (after hard decision is applied to each bits).
4: **if** $H\boldsymbol{c}'^T = \boldsymbol{0}$ **then**
5: **return** $\boldsymbol{c}'$.
6: **end if**
7: $\boldsymbol{c} \leftarrow \boldsymbol{c}'$.
8: $i \leftarrow i + 1$.
9: **end while**
10: **return** decoding failure.

We consider the case $H\boldsymbol{c}^T \neq 0$. Put the difference between noiseless received information $\boldsymbol{mG'}$ and decoding result of SPA (after hard decision to each bits) as $d_h(\boldsymbol{c}')$: i.e. $d_h(\boldsymbol{c}') = w(\boldsymbol{mG'} - \boldsymbol{c}')$. Thus $d_h(\boldsymbol{c}') - t$ represents decrease (if $d_h(\boldsymbol{c}') - t < 0$) and increase (if $d_h(\boldsymbol{c}') - t > 0$) of error bits number $\|\boldsymbol{e}\|_1$ after hard decision. Figure 3 show the number of observed values of $d_h(\boldsymbol{c}') - t$ when $t = 110$. For example, the number of trials which take $-5 \leq d_h(\boldsymbol{c}') - t \leq -1$ occupies 51, while the number of trials which take $1 \leq d_h(\boldsymbol{c}') - t \leq 5$ is 5, under total 100 decoding errors. Figure 4 shows the

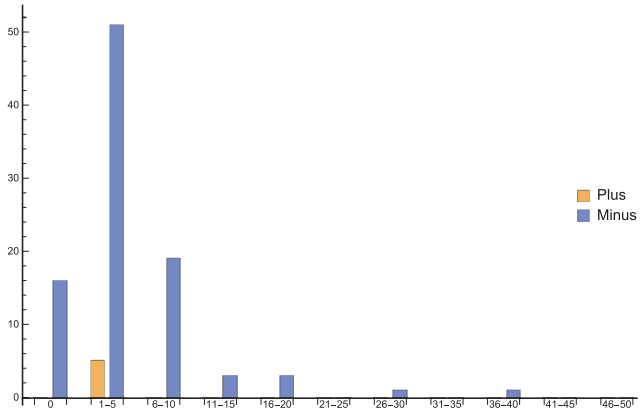

Fig. 3. The barchart of the number of observed values of $w(\boldsymbol{mG'} - h(\boldsymbol{c}'))$ in the case of SPA decoding and $(n, k, d_c, n_0) = (9602, 4801, 45, 2)$ code $(t = 110)$.

case $t = 95$. In this case, there is no case which takes $d_h(\boldsymbol{c}') - t \geq 0$. From Figure 3 and 4, we can expect that as the iteration number i increases, the value $d_h(\boldsymbol{c}') - t$ tends to decrease. As a result, all errors will be fixed as the iteration number increases. We note that the same effect cannot be expected for BF algorithm. Figure 5 and 6 are results obtained by replacing the decoding method SPA in Algorithm 4 to BF under the circumstance of Figure 3 and 4 respectively. We

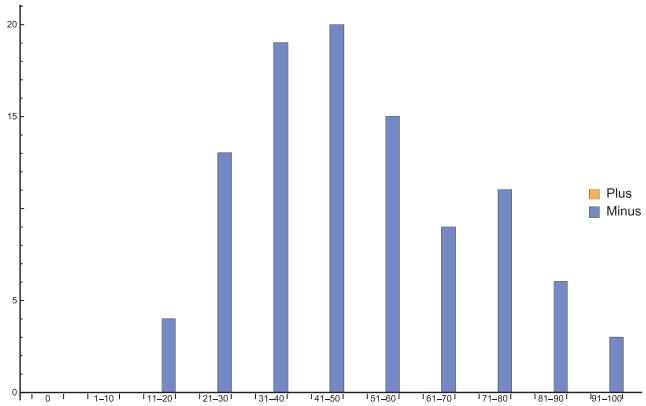

Fig. 4. The bar chart of the number of observed values of $w(\boldsymbol{mG'} - h(\boldsymbol{c}'))$ in the case of SPA decoding and $(n, k, d_c, n_0) = (9602, 4801, 45, 2)$ code $(t = 95)$.

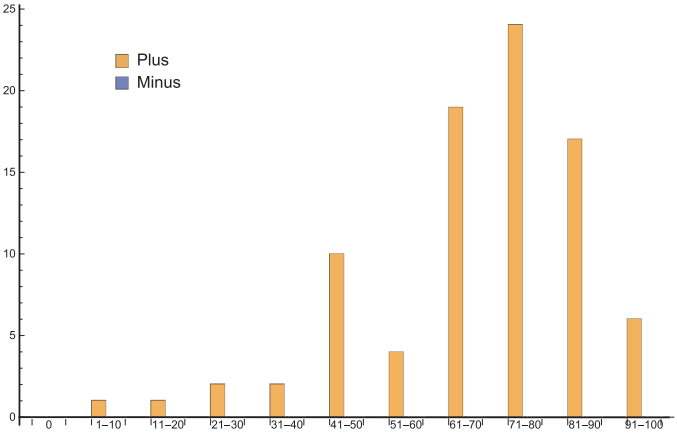

Fig. 5. The bar chart of the number of observed values of $w(\boldsymbol{mG'} - h(\boldsymbol{c}'))$ in the case of BF decoding and $(n, k, d_c, n_0) = (9602, 4801, 45, 2)$ code $(t = 110)$.

observe that Figure 5 and 6 show $d_h(\boldsymbol{c}') - t > 0$ almost all cases, and hence we cannot expect the reduction of erroneous bits by successive hard decision of decoding results.

After each iteration of RSPA, in most case, the number of erroneous bits is expected to reduce, hence by applying ADMM method to $\boldsymbol{c}$ obtained by hard decision in Step 7, further improvement in DER could be expected. We call this hybrid algorithm RSPA-ADMM method.

Algorithm 5 The RSPA-ADMM method.

1: **set** N_{SPA} as maximum iteration number, $i = 0$.
2: **while** $i < N_{\mathrm{SPA}}$ **do**
3: **set** $\boldsymbol{c}'$ as decoding result of SPA for $\boldsymbol{c}$ (after hard decision is applied to each bits).
4: **if** $H\boldsymbol{c}'^T = \boldsymbol{0}$ **then**
5: **return** $\boldsymbol{c}'$.
6: **end if**
7: $\boldsymbol{c} \leftarrow \boldsymbol{c}'$.
8: Apply ADMM decoding to $\boldsymbol{c}$. **set** $\boldsymbol{c}'$ as decoding result of ADMM for $\boldsymbol{c}$.
9: **if** $H\boldsymbol{c}'^T = \boldsymbol{0}$ **then**
10: **return** $\boldsymbol{c}'$.
11: **end if**

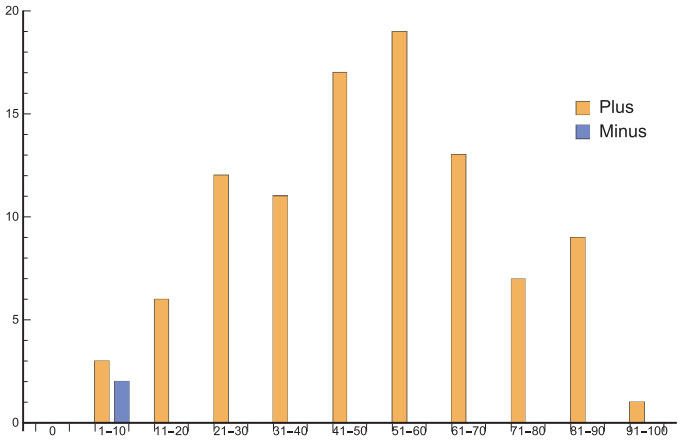

Fig. 6. The bar chart of the number of observed values of $w(\boldsymbol{mG'} - h(\boldsymbol{c'}))$ in the case of BF decoding and $(n, k, d_c, n_0) = (9602, 4801, 45, 2)$ code $(t = 95)$.

12: $i \leftarrow i + 1$.
13: **end while**
14: **return** decoding failure.

We note Step 8 can be done in parallel. As an effect of the introduction of RSPA-ADMM method, it is observed that the average decoding time is reduced to 1.55∼4.02 times compared to BF decoding. This is because in most cases, RSPA-ADMM algorithm ends with RSPA part.

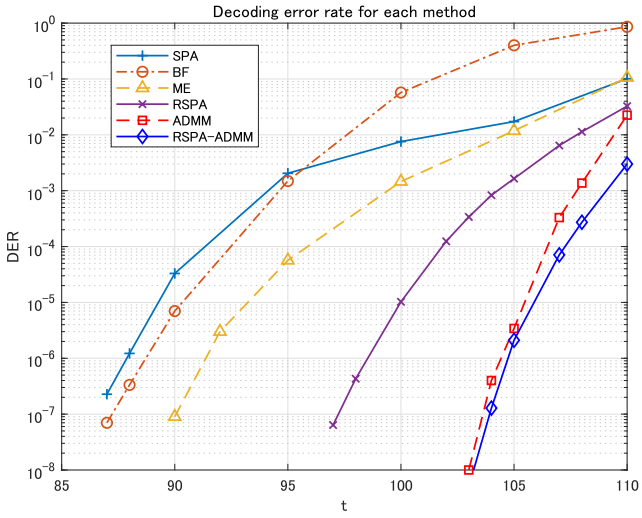

Fig. 7. DER of each decoding method for $(n, k, d_c, n_0) = (9602, 4801, 45, 2)$ code.

Figure 7 shows the DER of 6 methods (SPA, BF, ME, RSPA, ADMM, RSPA-ADMM) for $(n, k, d_c, n_0) = (9602, 4801, 45, 2)$ code. The maximum iteration number of SP was set to 50. The parameters (μ, α, M) in ADMM and RSPA-ADMM methods were fixed to $(6, 15, 10000)$. We note parameters μ and α were chosen by heuristic trials, hence these may not be an optimal choice. In spite of this, it holds that ADMM and RSPA-ADMM attains DFR lower than 10^{-7} at $t = 103$ compared to the result that BF decoding requires $t = 87$ for attaining almost the same DFR. We note DER of RSPA-ADMM when $t = 103$ might be less than 10^{-8}, since no decoding error was observed under 10^8 trials. Figure 8

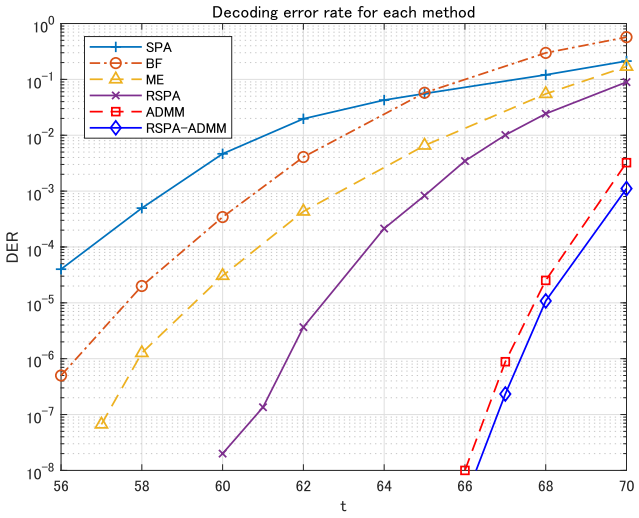

Fig. 8. DER of each decoding method for $(n, k, d_c, n_0) = (10779, 3593, 51, 3)$ code.

shows the DER of 6 methods (SPA, BF, ME, RSPA, ADMM, RSPA-ADMM) for $(n, k, d_c, n_0) = (10779, 3593, 51, 3)$ code. The maximum iteration number of SP was set to 50. The parameters (μ, α, M) in ADMM and RSPA-ADMM methods were fixed to $(8, 15, 10000)$. It seems to be remarkable that ADMM and RSPA-ADMM attains DFR lower than 10^{-7} at $t = 66$ compared to the result that BF decoding requires $t = 55$ for attaining almost the same DFR. We note DER of RSPA-ADMM when $t = 66$ might be less than 10^{-8}, since no decoding error was observed under 10^8 trials.

V. Discussion

This paper presented the effectiveness of ADMM based methods in decoding QC-MDPC codes. Since proposing two methods ADMM and RSPA-ADMM have abilities to correct more errors than standard error number assumed for BF algorithm, these ADMM based decoding can detect irregular usage of error numbers that aimed for the collection of DER information by adversaries. Hence, proposing schemes may serve as an audit device which guard the conventional BF algorithm based decoding systems.

Although ADMM based decoding showed certain improvements in DER, following problems remains. First, the systematic methods for the determination of the optimal parameters (μ, α) in ADMM decoding algorithm.

Second, an acceleration problem of the ADMM decoding methods. In Algorithm 2, Step 5 can be done in parallel. Thus, the acceleration of projection to $\Pi_{\mathbb{PP}_{n_0 d_c}}$ remains as essential problem. Zhang-Siegel [18] is proposing an alternative projection algorithm that is reported to be effective in the improvement of decoding speed for LDPC codes. The essential improvement of [18] is replacing sorting operation in Step 1 of Algorithm 3 with "cut search algorithm". Since the numbers of

$n_0 d_c$ that should be sorted in Step 1 of Algorithm 3 are rather large number for MDPC codes compared to LDPC codes cases, the "cut search algorithm" might work more effectively than LDPC codes cases in [18].

VI. Appendix

A. Preparation

In this section, we show the validity of projection algorithm 3 for the sake of completeness. The main difference with Algorithm 2 of [2] is the set $\hat{\boldsymbol{\beta}}$ in Step 11 of Algorithm 3. In Algorithm 2 of [2], the set $\hat{\boldsymbol{\beta}}$ is defined as:

$$\hat{\beta}_i = \begin{cases} v_i - 1, & (1 \le i \le r+1) \\ -v_i, & (r+2 \le i \le d). \end{cases}$$

This seems to be the main factor which cause non-negligible degradation of DER in Figure 1 and 2.

First, we prepare lemmas necessary for our purpose.

Lemma 6.1 ([2, Propostion 3]): Assume the components of $\boldsymbol{v} \in \mathbb{R}^d$ is sorted in descending (ascending) order, then the components of $\Pi_{\mathbb{PP}_d}$ maintain this order.

Definition 6.2: Assume the components of $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d$ are sorted in descending order, i.e.

$$v_1 \ge v_2 \ge \cdots \ge v_d, \quad w_1 \ge w_2 \ge \cdots \ge w_d.$$

Then the vector $\boldsymbol{w}$ is said to "majorizes" $\boldsymbol{u}$ if

$$\begin{cases} \sum_{k=1}^{q} u_k \le \sum_{k=1}^{q} w_k, & (1 \le q \le d-1) \\ \sum_{k=1}^{d} u_k = \sum_{k=1}^{d} w_k. \end{cases}$$

Lemma 6.3 ([2, Theorem 1]): Suppose vectors $\boldsymbol{u}$ and $\boldsymbol{w}$ are sorted in descending order. Then $\boldsymbol{u}$ is in the convex hull of all permutations of $\boldsymbol{w}$ if and only if $\boldsymbol{w}$ majorizes $\boldsymbol{u}$.

We define $\mathbb{P}_d^s = \{\boldsymbol{x} \in \{0,1\}^d \,|\, \|\boldsymbol{x}\|_1 = s\}$ and $\mathbb{PP}_d^s = \mathrm{conv}(\mathbb{P}_d^s)$. Recall that $\mathbb{PP}_d = \mathrm{conv}(\mathbb{P}_d)$, so it holds

$$\mathbb{PP}_d = \mathrm{conv}\left(\cup_{0 \le s \le d, s:\mathrm{even}} \mathbb{PP}_d^s\right). \tag{15}$$

From Lemma 6.3, it holds that $\boldsymbol{u} \in \mathbb{PP}_d^s \iff$

$$\begin{cases} \sum_{k=1}^{q} u_k \le \min(q, s), & (1 \le q \le d-1) \\ \sum_{k=1}^{d} u_k = s. \end{cases} \tag{16}$$

Further, following lemma (two-slice lemma) holds:

Lemma 6.4 ([2, Lemma 2]): Suppose the components of $\boldsymbol{z} \in \mathbb{PP}_d$ is sorted in descending order. Define r be the minimum even integer greater than or equal to $\lfloor \|\boldsymbol{z}\|_1 \rfloor_{\mathrm{even}}$. Then $\boldsymbol{z}$ is expressed by convex combination of $\mathbb{PP}_d^r$ and $\mathbb{PP}_d^{r+2}$.

Suppose $\boldsymbol{u} \in \mathbb{PP}_d^r \cap [0,1]^d$ and $\boldsymbol{v} \in \mathbb{PP}_d^{r+2} \cap [0,1]^d$. Then, from Lemma 6.4, for α satisfying $0 \le \alpha \le 1$, it holds

$$\begin{aligned} &\sum_{k=1}^{q} z_k = \sum_{k=1}^{q} (\alpha u_k + (1-\alpha) v_k) \qquad (17) \\ =&\alpha \sum_{k=1}^{q} u_k + (1-\alpha) \sum_{k=1}^{q} v_k \\ \le&\alpha \min(q, r) + (1-\alpha) \min(q, r+2), \quad (1 \le q \le d-1). \end{aligned}$$

and

$$\begin{aligned} &\sum_{k=1}^{d} z_k = \sum_{k=1}^{d} (\alpha u_k + (1-\alpha) v_k) \qquad (18) \\ =&\alpha \sum_{k=1}^{d} u_k + (1-\alpha) \sum_{k=1}^{d} v_k = \alpha r + (1-\alpha)(r+2). \end{aligned}$$

We note in the case $q \le r$, (17) clearly holds. While in the case $q \ge r+2$, (since $d-1 \ge q \ge r+2$, this case occurs when $r+3 \le d$), we can observe (17) holds if(18) is valid. Thus, (17) should be considered only for the case $q = r+1$. For the case $q = r+1$, (since $d-1 \ge q = r+1$, this case occurs when $r+2 \le d$) (17) becomes as follows

$$\sum_{k=1}^{r+1} z_k \le \alpha r + (1-\alpha)(r+1) = r+1-\alpha, \quad (r+2 \le d). \tag{19}$$

From (18), we have

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

and hence the projection $\Pi_{\mathbb{PP}_d}(\boldsymbol{v})$ for $\boldsymbol{v} \in \mathbb{R}^d$ is described as convex quadratic programming problem:

$$\begin{aligned} &\text{(P) Minimize} \quad \|\boldsymbol{v} - \boldsymbol{z}\|_2^2 && (20) \\ &\text{Subject to} \\ &\quad 0 \le z_i \le 1, \quad (1 \le i \le d) && (21) \\ &\quad z_i \ge z_{i+1}, \quad (1 \le i \le d-1) && (22) \\ &\quad r \le \sum_{k=1}^{d} z_k \le r+2 \leftrightarrow 0 \le \alpha \le 1 && (23) \\ &\quad \sum_{k=1}^{r+1} z_k - \sum_{k=r+2}^{d} z_k \le r, \quad (r+2 \le d) \leftrightarrow (19). && (24) \end{aligned}$$

We note (24) is valid only in the case $r+2 \le d$ and in this case we can express (24)by

$$\boldsymbol{f}_r^T \boldsymbol{z} \le r \tag{25}$$

where $\boldsymbol{f}_r^T = (1, \ldots, 1, -1, \ldots, -1)$.

Following lemma is well-known.

Lemma 6.5: Assume f, g_i, $(1 \le i \le m)$ are all convex functions on $\mathbb{R}^d$. Define the minimization problem:

$$\begin{aligned} &\text{(CP) Minimize} \quad f(x) && (26) \\ &\text{Subject to} \quad g_i(x) \le 0, \ (1 \le i \le m), \quad x \in \mathbb{R}^d. \end{aligned}$$

Suppose $\bar{x}$ satisfies Karush-Kuhn-Tucker (KKT) conditions. Then $\bar{x}$ is a (global) minimizer of problem (CP).

In the case of problem (P), f is a strictly convex function on $\mathbb{R}^d$, thus we obtain:

Lemma 6.6: Assume $\bar{x}$ satisfies KKT condition for problem (P). Then $\bar{x}$ is a unique (grobal) minimizer of problem (P).

B. Varidity of Algorithm 3

Since $r = \lfloor \|\hat{\boldsymbol{z}}\|_1 \rfloor_{\text{even}}$ (step 2), we have

$$r \leq d. \tag{27}$$

As noted (25) is valid only in the case $r+2 \leq d$, we consider the cases $r = d$ or $r+1 = d$ for problem (P). For these cases, from the definition of r it holds

$$r \leq \sum_{k=1}^{d} \hat{z}_k \leq r+2.$$

Since $v_1 \geq v_2 \geq \cdots \geq v_d$, $\hat{z}_k$ $(1 \leq k \leq d)$ keeps this order. Thus

$$\hat{z}_{i+1} - \hat{z}_k \geq 0, \quad (1 \leq k \leq d-1).$$

From the definition $0 \leq \hat{z}_k \leq 1$, $(1 \leq k \leq d)$. Hence $\hat{\boldsymbol{z}}$ is a minimizer of problem (P) under constraint conditions (21), (22) and (23); (see step 4 and 5).

Here after, we fix $r+2 \leq d$. First we consider the case $\boldsymbol{f}_r^T \hat{\boldsymbol{z}} \leq r$. As in the case $r = d$ or $r+1 = d$, $\hat{\boldsymbol{z}}$ satisfies conditions (21), (22) and (23). Moreover, form the assumption (24) is satisfied. Hence $\hat{\boldsymbol{z}}$ is a minimizer of problem (P); (see step 7 and 8).

Hence we can assume

$$\boldsymbol{f}_r^T \hat{\boldsymbol{z}} > r. \tag{28}$$

The KKT condition for the problem (P) is described as: there exists $\beta, \boldsymbol{\nu}, \boldsymbol{\eta}, \xi, \boldsymbol{\theta}$ and ζ such that following conditions are satisfied.

$$\begin{aligned}
\boldsymbol{z} =& \boldsymbol{v} - \beta \boldsymbol{f}_r - \boldsymbol{\nu} + \boldsymbol{\eta} - (\xi - \zeta)\mathbf{1} + S^T \boldsymbol{\theta}, && (29)\\
& 0 \leq \beta \perp \boldsymbol{f}_r^T \boldsymbol{z} - r \leq 0, && (30)\\
& \mathbf{0} \leq \boldsymbol{\nu} \perp \boldsymbol{z} \leq \mathbf{1}, && (31)\\
& \mathbf{0} \leq \boldsymbol{\eta} \perp \boldsymbol{z} \geq \mathbf{0}, && (32)\\
& \boldsymbol{\theta} \perp S\boldsymbol{z} \geq 0, && (33)\\
& 0 \leq \xi \perp \mathbf{1}^T \boldsymbol{z} - r - 2 \leq 0, && (34)\\
& 0 \leq \zeta \perp \mathbf{1}^T \boldsymbol{z} - r \geq 0, && (35)
\end{aligned}$$

then $\boldsymbol{z}$ is a unique minimizer of (P). Here $\perp$ means for example for (31) $\nu_i(1 - z_i) = 0$, $(1 \leq i \leq d)$ and

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

We note the uniqueness of the minimizer follows from Lemma 6.5.

In the following we see that such $\beta, \boldsymbol{\nu}, \boldsymbol{\eta}, \xi, \boldsymbol{\theta}$ and ζ exists. For fixed $\beta > 0$ we decide ν_i, $(1 \leq i \leq d)$ as follows:

$$\nu_i = \begin{cases} 0, & v_i - \beta(\boldsymbol{f}_r)_i \leq 1 \\ v_i - \beta((\boldsymbol{f}_r)_i) - 1, & v_i - \beta(\boldsymbol{f}_r)_i > 1. \end{cases} \tag{36}$$

Similarly, η_i is decided as follows:

$$\eta_i = \begin{cases} 0, & v_i - \beta(\boldsymbol{f}_r)_i \geq 0 \\ -v_i + \beta(\boldsymbol{f}_r)_i, & v_i - \beta(\boldsymbol{f}_r)_i < 0. \end{cases} \tag{37}$$

Moreover, we fix $\xi = \zeta = 0$ and $\boldsymbol{\theta} = \mathbf{0}$. From these $\boldsymbol{z}$, which is the LHS of (29) is regarded as a function of β and expressed as

$$\boldsymbol{z}(\beta) = \Pi_{[0,1]^d}(\boldsymbol{v} - \beta \boldsymbol{f}_r). \tag{38}$$

Hence, from (36), condition (31) is satisfied, also form (37), condition (32) is satisfied. As Step 10, range of β is defined as

$$0 \leq \beta \leq \beta_{\max} = \frac{1}{2}(v_{r+1} - v_{r+2}).$$

We note following holds:

$$\beta \leq \beta_{\max} \longleftrightarrow v_{r+2} + \beta \leq v_{r+1} - \beta.$$

From

$$\begin{aligned}
&(\boldsymbol{z}(\beta_{\max}))_{r+1} = \Pi_{[0,1]}(v_{r+1} - \beta_{\max}) \\
&= \Pi_{[0,1]}\left(\frac{1}{2}(v_{r+1} + v_{r+2})\right)
\end{aligned}$$

and

$$\begin{aligned}
&(\boldsymbol{z}(\beta_{\max}))_{r+2} = \Pi_{[0,1]}(v_{r+2} + \beta_{\max}) \\
&= \Pi_{[0,1]}\left(\frac{1}{2}(v_{r+1} + v_{r+2})\right) = (\boldsymbol{z}(\beta_{\max}))_{r+1},
\end{aligned}$$

we have

$$\begin{aligned}
\boldsymbol{f}_r \boldsymbol{z}(\beta_{\max}) =& \sum_{i=1}^{r} (\boldsymbol{z}(\beta_{\max}))_i + (\boldsymbol{z}(\beta_{\max}))_{r+1} \\
& - (\boldsymbol{z}(\beta_{\max}))_{r+2} - \sum_{i=r+3}^{d} (\boldsymbol{z}(\beta_{\max}))_i \\
=& \sum_{i=1}^{r} (\boldsymbol{z}(\beta_{\max}))_i - \sum_{i=r+3}^{d} (\boldsymbol{z}(\beta_{\max}))_i \\
\leq& r - \sum_{i=r+3}^{d} (\boldsymbol{z}(\beta_{\max}))_i \leq r.
\end{aligned} \tag{39}$$

Since from (28)

$$\boldsymbol{f}_r \boldsymbol{z}(0) = \boldsymbol{f}_r \hat{\boldsymbol{z}} > r,$$

there exists $\beta_{\max}$ $(0 < \beta_{\text{opt}} \leq \beta_{\max})$ such that

$$\boldsymbol{f}_r \boldsymbol{z}(\beta_{\text{opt}}) = r. \tag{40}$$

We will show that $\boldsymbol{z}(\beta_{\text{opt}})$ satisfies KKT condition (recall that conditions (31) and (32) are already satisfied). From (40), the condition (30) is satisfied.

To see (33), $S\boldsymbol{z}(\beta_{\text{opt}}) \geq 0$ is enough (since $\boldsymbol{\theta} = 0$, $\perp$ relation is satisfied). From the assumption (Step 1), for non-negative β it holds

$$v_1 - \beta \geq v_2 - \beta \geq \cdots \geq v_{r+1} - \beta$$

and

$$v_{r+2} + \beta \geq v_{r+3} + \beta \geq \cdots \geq v_d + \beta.$$

Since

$$v_{r+1} - \beta \geq v_{r+2} + \beta \longleftrightarrow \beta \leq \frac{1}{2}(v_{r+1} - v_{r+2}) = \beta_{\max},$$

we have

$$(\boldsymbol{z}(\beta_{\text{opt}}))_1 \geq (\boldsymbol{z}(\beta_{\text{opt}}))_2 \geq \cdots \geq (\boldsymbol{z}(\beta_{\text{opt}}))_{r+1} \geq (\boldsymbol{z}(\beta_{\text{opt}}))_{r+2} \geq \cdots \geq (\boldsymbol{z}(\beta_{\text{opt}}))_d.$$

Thus, we have shown (33).

To see (34) and (35)

$$r \leq \sum_{i=1}^{d} (\boldsymbol{z}(\beta_{\text{opt}}))_i \leq r+2 \tag{41}$$

is enough. From (28),

$$r \leq r+2 \sum_{i=r+2}^{d} (\boldsymbol{z}(\beta_{\text{opt}}))_i = \sum_{i=1}^{d} (\boldsymbol{z}(\beta_{\text{opt}}))_i.$$

Also from (28),

$$\sum_{i=r+2}^{d} (\boldsymbol{z}(\beta_{\text{opt}}))_i = \sum_{i=1}^{r+1} (\boldsymbol{z}(\beta_{\text{opt}}))_i - r \leq r+1-r = 1,$$

we obtain

$$\sum_{i=1}^{d} (\boldsymbol{z}(\beta_{\text{opt}}))_i = \sum_{i=1}^{r+1} (\boldsymbol{z}(\beta_{\text{opt}}))_i + \sum_{i=r+2}^{d} (\boldsymbol{z}(\beta_{\text{opt}}))_i \leq r+1+1 = r+2.$$

Hence, we have shown conditions (34) and (35). Moreover, from Lemma 6.5, β_{opt} is unique. Summarizing the argument, we have shown that $\boldsymbol{z}(\beta_{\text{opt}})$ is the unique minimizer of the problem (P).

C. Searching method for β_{opt}

Since β_{opt} is unique, bi-section method is effective. Define

$$\varphi(\beta) = \boldsymbol{f}_r \boldsymbol{z}(\beta).$$

We can observe that φ is continuous and piecewise linear function. The possible non-differentiable points are such β, satisfying $v_i - \beta = 1$ or $v_i - \beta = 0$ or $v_i + \beta = 1$ or $v_i + \beta = 0$. We divide set of these possible non-differentiable points in to four sets:

$$\begin{aligned}
\mathcal{E}_1 &= \{v_i - 1 \,|\, 1 \leq i \leq r+1\}, \\
\mathcal{L}_1 &= \{v_i \,|\, 1 \leq i \leq r+1\}, \\
\mathcal{E}_2 &= \{1 - v_i \,|\, r+2 \leq i \leq d\}, \\
\mathcal{L}_2 &= \{v_i \,|\, 1 \leq r+2 \leq d\}
\end{aligned}$$

and put

$$\hat{\boldsymbol{\beta}} = \mathcal{E}_1 \cup \mathcal{L}_1 \cup \mathcal{E}_2 \cup \mathcal{L}_2.$$

We set

$$\tilde{\boldsymbol{\beta}} \leftarrow \left\{ \tilde{\beta} \in \hat{\boldsymbol{\beta}} \cup \{0\} \cup \{\beta_{\max}\} \,|\, 0 \leq \tilde{\beta} \leq \beta_{\max} \right\}$$

and sort $\hat{\boldsymbol{\beta}}$ to $\boldsymbol{\beta}$ in ascending order. Hence $\boldsymbol{\beta} = \{\beta_1, \beta_2, \ldots \beta_p\}$ satisfies $0 = \beta_0 < \beta_1 < \cdots < \beta_p = \beta_{\max}$ (Step 12 and 13). We can find β_{opt} effectively with bi-section method by finding points β_j and β_{j+1} satisfying $\sum_{i=1}^{s} (\boldsymbol{f}_r)_i (\boldsymbol{z}(\beta_j))_i > r$ and $\sum_{i=1}^{s} (\boldsymbol{f}_r)_i (\boldsymbol{z}(\beta_{j+1}))_i > r$ $(0 \leq j \leq p-1)$.

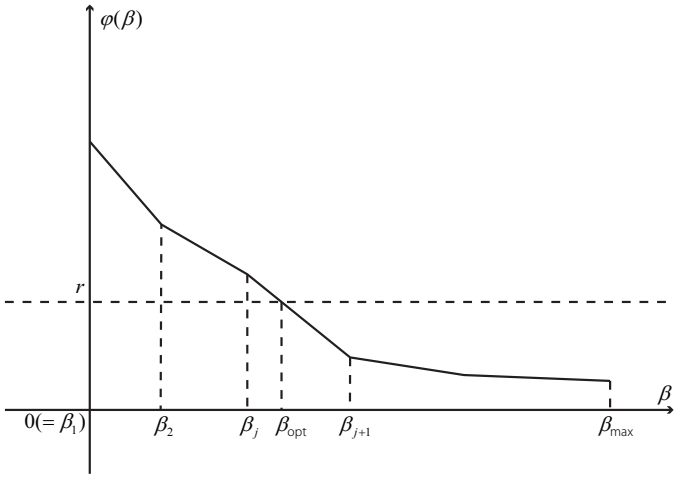

Fig. 9. The graph of φ.

REFERENCES

[1] M. Baldi, *QC-LDPC Code-Based Cryptography*, Springer 2014.

[2] S. Barman, X. Liu, S. C. Draper and B. Recht, "Decomposition Methods for Large Scale LP Decoding," IEEE Transactions on Information Theory, 59, 12, pp. 7870-7886, 2013.

[3] E. Berlekamp, R. McEliece and H. van Tilborg, "On the inherent intractability of certain coding problems," IEEE Transactions on Information Theory, 24, 3, pp. 384-386, 1978.

[4] *BIKE (Bit Flipping Key Encapsulation)*, Available: https://bikesuite.org

[5] I. E. Bocharova, T. Johansson and B. D. Kudryashov, "Improved iterative decoding of QC-MDPC codes in the McEliece public key cryptosystem," 2019 IEEE International Symposium on Information Theory (ISIT), pp. 1882-1886, 2019.

[6] N. Drucker, S. Gueron, D. Kostic, "QC-MDPC Decoders with Several Shades of Gray", In: Ding, J., Tillich, JP. (eds) Post-Quantum Cryptography. PQCrypto 2020. Lecture Notes in Computer Science, vol. 12100. Springer, 2020

[7] J. Feldman, M. J. Wainwright and D. R. Karger, "Using linear programming to decode binary linear codes," IEEE Transactions on Information Theory, 51, 3, pp. 954-972, 2005.

[8] Q. Guo, T. Johansson and P. Stankovski Wagner, "A Key Recovery Reaction Attack on QC-MDPC," IEEE Transactions on Information Theory, 65, 3, pp. 1845-1861, 2019.

[9] Gurobi Optimizer, https://www.gurobi.com/

[10] H. Kaneko, "Look-Ahead Bit-Flipping Decoding of MDPC Code", 2022 IEEE International Symposium on Information Theory (ISIT), pp. 2922-2927, 2022.

[11] X. Liu and S. C. Draper, "The ADMM Penalized Decoder for LDPC Codes," IEEE Transactions on Information Theory, 62, 6, pp. 2966-2984, 2016.

[12] R. Misoczki, J. Tillich, N. Sendrier and P. S. L. M. Barreto, "MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes", 2013 IEEE International Symposium on Information Theory, pp. 2069-2073, 2013.

[13] A. Nilsson, I. E. Bocharova, B. D. Kudryashov and T. Johansson, "A Weighted Bit Flipping Decoder for QC-MDPC-based Cryptosystems," 2021 IEEE International Symposium on Information Theory (ISIT), pp. 1266-1271 2021.

[14] N. Sendrier and V. Vasseur, "About Low DFR for QC-MDPC Decoding", in *International Conference on Post-Quantum Cryptography 2020*, Lecture Notes in Computer Science, 12100, 2020.

[15] QC-MDPC decoder, https://github.com/vvasseur/qcmdpc_decoder

[16] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring", Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124-134, 1994.

[17] J. Stern, "A method for finding codewords fo small weight" in *Coding theory and applications*, Lecture Notes in Computer Science, 388, 1989.

[18] X. Zhang and P. H. Siegel, "Efficient Iterative LP Decoding of LDPC Codes with Alternating Direction Method of Multipliers", 2013 IEEE International Symposium on Information Theory, pp. 1501-1505, 2013.

[19] *HQC (Hamming Quasi-Cyclic)* , Available: https://pqc-hqc.org

[20] Post-Quantum Cryptography, https://csrc.nist.gov/Projects/post-quantum-cryptography/