Zero-shot deep-unfolding decoder for QC-MDPC McEliece cryptosystems

Shingo Kukita[1][0000-0003-1526-4570], Rei Iseki[1], Takeshi Namatame[1], and Kohtaro Watanabe[1][0000-0001-9674-0924]

Department of Computer Science, National Defense Academy of Japan, Kanagawa, Japan
{kukita,wata}@nda.ac.jp

Abstract. **Keywords:** Code-based cryptography · QC-MDPC McEliece cryptosystem · Deep learning.

The QC-MDPC McEliece cryptosystem is a promising candidate for post-quantum cryptography, and the decoding performance of the underlying QC-MDPC code directly affects the security of the scheme. Deep unfolding, a framework that unfolds an iterative algorithm into a neural network with trainable weights, has been shown to improve belief propagation (BP) decoding for codes with dense parity-check matrices. However, applying deep unfolding directly to the large QC-MDPC codes used in practice is impractical owing to the computational cost of training. Moreover, in QC-MDPC-based cryptosystems, the parity-check matrix serves as the secret key and must be replaced periodically; key-specific training would therefore need to be repeated at each replacement. We address both issues through zero-shot transfer. We propose weight homogenisation, which constrains the trainable weights to a single scalar per iteration, making them independent of the specific Tanner graph. This enables a decoder trained on a small QC-MDPC code to be applied directly to larger codes. Experiments on QC-MDPC codes with parameters proposed for 80-bit and 128-bit security demonstrate that the proposed method achieves a lower decoding error rate than standard BP.

1 Introduction

Public key cryptography [6] underpins the security of modern networked society. The security of a public key cryptosystem relies on the computational intractability of certain mathematical problems; for example, the RSA cryptosystem [21], one of the most widely deployed, is founded on the difficulty of integer factorisation. However, Shor demonstrated that a sufficiently large quantum computer could efficiently solve this problem, posing a potential threat to RSA and related cryptosystems [23]. This threat has motivated extensive research into post-quantum cryptography, i.e., cryptographic schemes that remain secure against quantum adversaries. Candidate post-quantum schemes are typically categorised by the underlying mathematical problem, including lattice-based, code-based, hash-based, and multivariate approaches [4]. In 2024, following a multi-year evaluation process, NIST published its first three post-quantum cryptographic standards, primarily based on lattice problems [16–19].

The McEliece cryptosystem [12] is a code-based cryptosystem whose security relies on the difficulty of decoding random linear codes [3, 10], and has long been studied in the context of post-quantum cryptography. The original scheme, which employs Goppa codes, offers strong security guarantees but requires a large generator matrix as a public key, limiting its practicality. To address this limitation, a variant based on quasi-cyclic moderate-density parity-check (QC-MDPC) codes was proposed [13]; the quasi-cyclic structure of the code allows the generator matrix to be reconstructed from a compact representation, reducing the public key size. BIKE is a key encapsulation mechanism built on QC-MDPC codes that was evaluated in the fourth round of the NIST post-quantum cryptography standardisation process [2]. Although NIST ultimately selected HQC for standardisation in 2025 [1], BIKE remains an active subject of research owing to its compact key sizes.

In a code-based cryptosystem, the decoding error rate (DER) of the underlying code directly affects the security of the scheme. For instance, to achieve IND-CCA security at the 128-bit security level, the DER is required to be below 2^{-128} [2,22]. A low DER is also required for resistance against attacks such as the GJS reaction attack on the QC-MDPC McEliece cryptosystem, which exploits variations in DERs across message types to recover secret key information [8].

Several methods have been employed to decode QC-MDPC codes, including bit-flipping algorithms [5, 20], ADMM [11, 25], and belief propagation (BP) [7]. BP decoding was introduced for low-density parity-check (LDPC) codes, where it achieves maximum-likelihood decoding when the Tanner graph associated with the parity-check matrix is cycle-free [7, 24]. For QC-MDPC codes, however, BP does not in general achieve maximum-likelihood performance, as the parity-check matrix is denser than that of typical LDPC codes. The message-passing operations in BP are soft-valued and hence differentiable, making BP amenable to *deep unfolding* [9,14,15], a deep-learning-based approach in which the iterative decoding process is unfolded into a layered neural network and the parameters at each iteration are optimised via training. Nachmani et al. [14] demonstrated that this approach improves BP decoding for codes with dense parity-check matrices, such as BCH codes. Bit-flipping algorithms, by contrast, rely on hard-decision steps that are not naively amenable to gradient-based optimisation.

In this paper, we apply deep unfolding to BP decoding of the large-size QC-MDPC codes used in code-based cryptosystems. A direct application of deep unfolding to such codes is impractical: the computational cost of training grows with the number of edges in the Tanner graph, which is prohibitively large for practical QC-MDPC codes. Moreover, as the Tanner graph depends on the parity-check matrix, which serves as the secret key and must be changed periodically, a decoder trained for one key cannot be reused for another. To overcome these difficulties, we constrain the trainable parameters to be uniform within each decoding iteration, reducing the number of free parameters to one per iteration. This design makes the decoder independent of the specific code structure, enabling a decoder trained on a small QC-MDPC code to be applied directly to larger codes without retraining. We demonstrate that this zero-shot transfer yields improved decoding performance for large-size QC-MDPC codes.

The remainder of this paper is organised as follows. Section 2 introduces the notation and fundamentals used throughout the paper. Section 3 reviews BP decoding. Section 4 describes the deep unfolding method and applies it to construct a decoder for small-size QC-MDPC codes, whose performance is then

evaluated. Section 5 extends the approach to large-size QC-MDPC codes and evaluates the resulting decoder. Section 6 provides a discussion and summary.

2 Preliminaries

A binary linear code of length n and dimension k is characterised by a parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{r\times n}$ and a generator matrix $\mathbf{G} \in \mathbb{F}_2^{k\times n}$, where $r = n - k$ denotes the number of parity-check equations and $\mathbb{F}_2$ is the binary field. The set of codewords is given by

$$C = \{\vec{x} \in \mathbb{F}_2^n \mid \mathbf{H}\vec{x} = \vec{0}\}. \tag{1}$$

We define the sets

$$\begin{aligned} A_i &:= \{j \in [n] \mid (\mathbf{H})_{ij} = 1\}, \quad i \in [r], \\ B_j &:= \{i \in [r] \mid (\mathbf{H})_{ij} = 1\}, \quad j \in [n], \end{aligned} \tag{2}$$

where $[n] := \{1, \ldots, n\}$. The set A_i collects the columns having a 1 in the ith row and B_j collects the rows having a 1 in the jth column.

The Tanner graph associated with $\mathbf{H}$ is a bipartite graph whose vertices are partitioned into n variable nodes and r check nodes. The jth variable node corresponds to the jth column of $\mathbf{H}$ and the ith check node to the ith row. The jth variable node and the ith check node are connected by an edge if and only if $(\mathbf{H})_{ij} = 1$. We denote the total number of such edges by E. See Fig. 1 for an illustration.

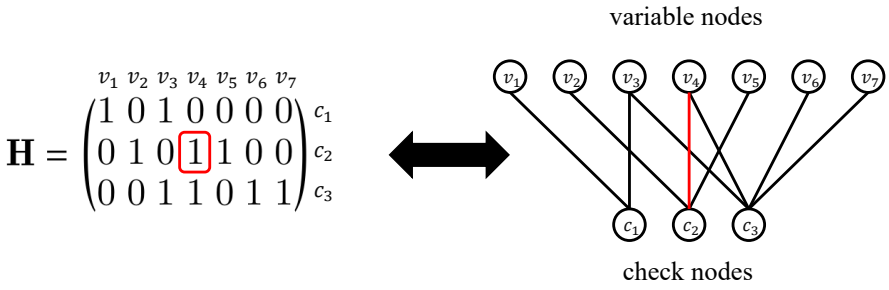


Fig. 1. A parity-check matrix and its Tanner graph. Variable nodes v_j correspond to columns and check nodes c_i to rows. The edge between v_4 and c_2 is highlighted in red.

A quasi-cyclic moderate-density parity-check (QC-MDPC) code [13] is a linear code whose parity-check matrix has the form

$$\mathbf{H} = [\mathbf{H}_0\ \mathbf{H}_1\ \cdots\ \mathbf{H}_{n_0-1}], \tag{3}$$

where each $\mathbf{H}_i$ $(0 \le i < n_0)$ is an $r \times r$ circulant matrix. The code length is $n = n_0 r$. The row weight of $\mathbf{H}$ is of order $O(\sqrt{n \log n})$, which is higher than that of typical LDPC codes. QC-MDPC codes are used to instantiate the McEliece cryptosystem [12]; in this QC-MDPC McEliece cryptosystem, the parity-check matrix serves as the secret key and must be replaced periodically.

3 Belief propagation

This section reviews the BP decoding algorithm. BP is a standard decoding algorithm for LDPC codes [7] and achieves maximum-likelihood decoding when the associated Tanner graph is cycle-free [24]. For QC-MDPC codes, however, the parity-check matrix is denser, resulting in numerous cycles in the Tanner graph; BP therefore no longer guarantees maximum-likelihood performance.

A codeword $\vec{x} = (x_1, \ldots, x_n) \in C$ is transmitted over a memoryless binary symmetric channel (BSC), which produces a received word $\vec{y} = (y_1, \ldots, y_n)$. The memoryless assumption gives

$$P(\vec{y} \mid \vec{x}) = \prod_{j \in [n]} P(y_j \mid x_j). \tag{4}$$

The BP algorithm estimates the transmitted word $\vec{x}$ by computing the marginal posterior probability $P(x_j \mid \vec{y})$ for each $j \in [n]$ from the received word $\vec{y}$ and the parity-check matrix $\mathbf{H}$. To this end, we define the log channel ratio vector

$$\vec{\lambda} = (\lambda_1, \ldots, \lambda_n), \quad \lambda_j = \ln \frac{P(y_j \mid 0)}{P(y_j \mid 1)}, \quad j \in [n]. \tag{5}$$

We assign a label $e \in [E]$ to each edge of the Tanner graph and write $\psi_v(e)$ and $\psi_c(e)$ for the variable and check nodes incident to edge e, respectively. We record which node is incident to each edge via two binary matrices $\mathbf{U} \in \mathbb{F}_2^{n \times E}$ and $\mathbf{V} \in \mathbb{F}_2^{r \times E}$, defined by

$$(\mathbf{U})_{j,e} = \delta_{j,\psi_v(e)}, \quad (\mathbf{V})_{i,e} = \delta_{i,\psi_c(e)}. \tag{6}$$

Introducing two message vectors $\vec{\alpha}, \vec{\beta} \in \mathbb{R}^E$, the BP algorithm is summarised in Algorithm 1.

Algorithm 1 The BP decoding

Input: max iteration L, connection matrices $\mathbf{U}$ and $\mathbf{V}$, log channel ratio vector $\vec{\lambda}$, initialised message vector $\vec{\alpha} = \vec{0}$.
Output: The estimated codeword $\vec{x}$, or decoding failure $\perp$.

1: **for** l $(1 \le l \le L)$ **do**
2: $\quad \vec{\beta} = \mathbf{U}^{\mathrm{T}}\vec{\lambda} + (\mathbf{U}^{\mathrm{T}}\mathbf{U} - \mathbf{I})\,\vec{\alpha}$.
3: $\quad \vec{\alpha}_{\mathrm{abs}} = 2\tanh^{-1}\left(\exp\left((\mathbf{V}^{\mathrm{T}}\mathbf{V} - \mathbf{I})\ln\left|\tanh\frac{\vec{\beta}}{2}\right|\right)\right)$.
4: $\quad \mathrm{sign}\vec{\alpha} = \left(\vec{1} - 2\mathbf{V}^{\mathrm{T}} \bmod_2 \left(\frac{\mathbf{V}\left(\vec{1} - \mathrm{sign}\vec{\beta}\right)}{2}\right)\right) \odot \mathrm{sign}\vec{\beta}$.
5: $\quad \vec{\alpha} = \mathrm{sign}\vec{\alpha} \odot \vec{\alpha}_{\mathrm{abs}}$.
6: $\quad \vec{\gamma} = \vec{\lambda} + \mathbf{U}\vec{\alpha}$.
7: $\quad \vec{z} = \frac{\vec{1} + \mathrm{sign}\vec{\gamma}}{2}$.
8: $\quad$ **if** $\mathbf{H}\vec{z} = \vec{0}$ **then**
9: $\quad\quad$ **return** z as the estimated codeword.
10: $\quad$ **end if**
11: **end for**

12: **return** $\perp$

In Algorithm 1, $\vec{1}$ denotes the all-one vector, whose dimension is E on line 4 and n on line 7. The function mod_2 returns its argument reduced modulo 2 and acts elementwise on vectors. The vector $\vec{\gamma}$ is called the log-likelihood ratio (LLR) vector. The derivation of the vectorised form is given in Appendix A.

4 Deep unfolding

The iterative structure of the BP algorithm lends itself to interpretation as a deep learning model: each iteration is unfolded as a layer of a neural network whose structure mirrors the BP update rule and whose edges carry trainable weights (Fig. 2). By optimising these weights using training data, the decoding performance can be improved. This framework is known as *deep unfolding* [9]. Nachmani et al. [14,15] applied this framework to BP decoding and demonstrated improved performance for codes with dense parity-check matrices. There are various ways to assign the weights, and we collectively refer to such decoders as deep unfolding decoders (DUDs).

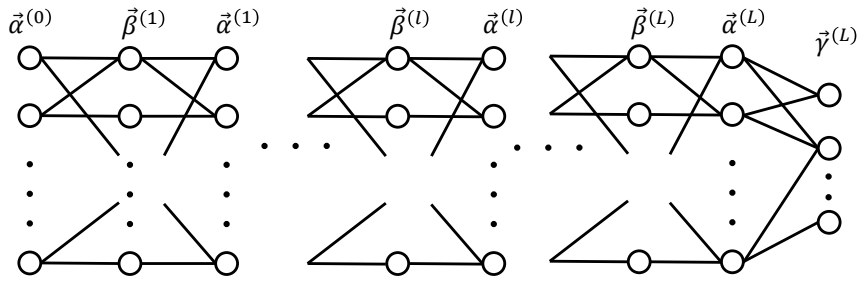


Fig. 2. The BP algorithm unfolded as a layered network. Superscripts in parentheses indicate the iteration index. Edges between layers represent data dependencies in the update rule. The LLR vector $\vec{\gamma}^{(l)}$ is computed at every iteration but only $\vec{\gamma}^{(L)}$ is shown.

Throughout this paper, we consider DUDs in which trainable weights are applied to $\vec{\beta}$; we call these β-weighted DUDs. This approach is similar to that of Nachmani et al. [14], though our formulation is simpler; see [14] for the full construction. In a β-weighted DUD, after computing $\vec{\beta}^{(l)}$ at the lth iteration (line 2 in Algorithm 1), we apply an elementwise scaling

$$\vec{\beta}'^{(l)} = \vec{\xi}^{(l)} \odot \vec{\beta}^{(l)}, \tag{7}$$

where $\vec{\xi}^{(l)} = (\xi_1^{(l)}, \xi_2^{(l)}, \ldots, \xi_E^{(l)}) \in \mathbb{R}^E$ is the weight vector at iteration l.

We now describe the learning procedure. We fix a parity-check matrix $\mathbf{H}_{\mathrm{learn}}$ of a QC-MDPC code for training. From a random message vector $\vec{m} \in \mathbb{F}_2^k$, we generate a codeword $\vec{x} = \mathbf{G}^T\vec{m}$, where $\mathbf{G}$ is the corresponding generator matrix.

An error vector $\vec{e}$ of fixed Hamming weight $w(\vec{e})$ is chosen uniformly at random, and the received word is obtained as $\vec{y} = \vec{x} + \vec{e}$. We prepare K such pairs $\{\vec{x}_p, \vec{y}_p\}_{p=1}^{K}$ as a training minibatch. Because the error weight is fixed, the bit flips in $\vec{e}$ are not independent and the channel transition probability $P(\vec{y} \mid \vec{x})$ does not factorise into a product of per-bit probabilities. BP nonetheless requires a per-bit channel LLR as input, so we approximate the channel by a memoryless BSC with crossover probability $w(\vec{e})/n$, giving

$$P(\vec{y} \mid \vec{x}) = \prod_{j \in [n]} P(y_j \mid x_j), \quad P(y_j \mid x_j) = \begin{cases} 1 - \dfrac{w(\vec{e})}{n}, & x_j = y_j, \\ \dfrac{w(\vec{e})}{n}, & x_j \neq y_j. \end{cases} \tag{8}$$

For each received word $\vec{y}_p$ in the minibatch, we run Algorithm 2 with mode = soft to obtain the soft estimate $\vec{z}_{\mathrm{soft}}(\vec{y}_p)$. The use of tanh in place of sign ensures differentiability with respect to the weights. We minimise the loss function

$$\Lambda = \sum_{p=1}^{K} \| \vec{z}_{\mathrm{soft}}(\vec{y}_p) - \vec{x}_p \|^2 \tag{9}$$

over $\{\vec{\xi}^{(l)}\}_{l=1}^{L}$ using the Adam optimiser. The above procedure is repeated L_{learn} times, each time with a freshly generated minibatch. To decode received words in practice, we run Algorithm 2 with the learned weights and mode = hard.

Algorithm 2 β-weighted DUD

Input: max iteration L, connection matrices $\mathbf{U}$ and $\mathbf{V}$, log channel ratio vector $\vec{\lambda}$, initialised message vector $\vec{\alpha} = \vec{0}$, weight vectors $\vec{\xi}^{(l)}$ for $l = 1, \ldots, L$, output mode $\in \{\mathrm{hard}, \mathrm{soft}\}$.

Output: Estimated codeword $\vec{z}$, soft estimate $\vec{z}_{\mathrm{soft}}$, or decoding failure $\perp$.

success $\leftarrow$ false.
for l $(1 \le l \le L)$ **do**
  $\vec{\beta} = \mathbf{U}^{\mathrm{T}} \vec{\lambda} + (\mathbf{U}^{\mathrm{T}} \mathbf{U} - \mathbf{I})\, \vec{\alpha}$.
  $\vec{\beta}' = \vec{\xi}^{(l)} \odot \vec{\beta}$.
  $\vec{\alpha}_{\mathrm{abs}} = 2 \tanh^{-1} \left(\exp\left((\mathbf{V}^{\mathrm{T}} \mathbf{V} - \mathbf{I}) \ln \left| \tanh \frac{\vec{\beta}'}{2} \right| \right) \right)$.
  $\mathrm{sign}\vec{\alpha} = \left(\vec{1} - 2\mathbf{V}^{\mathrm{T}} \bmod_2 \left(\frac{\mathbf{V}\left(\vec{1} - \mathrm{sign}\vec{\beta}'\right)}{2} \right) \right) \odot \mathrm{sign}\vec{\beta}'$.
  $\vec{\alpha} = \mathrm{sign}\vec{\alpha} \odot \vec{\alpha}_{\mathrm{abs}}$.
  $\vec{\gamma} = \vec{\lambda} + \mathbf{U}\vec{\alpha}$.
  $\vec{z} = \frac{\vec{1} + \mathrm{sign}\vec{\gamma}}{2}$.
  if $\mathbf{H}\vec{z} = \vec{0}$ **then**
    success $\leftarrow$ true.
    break.
  end if
end for
if mode = soft **then**
  return $\vec{z}_{\mathrm{soft}} = \frac{\vec{1} + \tanh \vec{\gamma}}{2}$.

17: **end if**
18: **if** success **then**
19: **return** $\vec{z}$.
20: **else**
21: **return** $\perp$.
22: **end if**

We propose two modifications to Algorithm 2). The first, weight homogenisation (HOM), replaces the elementwise scaling $\vec{\beta}' = \vec{\xi}^{(l)} \odot \vec{\beta}$ with a scalar multiplication

$$\vec{\beta}' = \xi^{(l)} \vec{\beta}, \tag{10}$$

reducing the number of trainable parameters from $L \cdot E$ to L. Because the scalar weights $\{\xi^{(l)}\}_{l=1}^{L}$ are independent of the specific Tanner graph, a decoder trained on one parity-check matrix can be applied directly to another without retraining. This is particularly suited to QC-MDPC-based cryptosystems, in which the parity-check matrix serves as the secret key and must be replaced periodically.

The second modification, LLR accumulation (ACC), replaces the LLR vector $\vec{\gamma}^{(l)}$ with the accumulated LLR

$$\vec{\Gamma}^{(l)} = \sum_{l'=1}^{l} \vec{\gamma}^{(l')}. \tag{11}$$

Accordingly, all occurrences of $\vec{\gamma}$ in Algorithm 2 are replaced by $\vec{\Gamma}^{(l)}$. The accumulation allows gradient information to reach the early iterations. HOM and ACC can be applied independently or in combination.

We compare four schemes: the original β-weighted DUD (Algorithm 2), HOM alone, ACC alone, and their combination HOM+ACC. The QC-MDPC code has parameters $(n, r) = (512, 256)$, row weight $w = 30$, and error weight $w(\vec{e}) = 13$, which are small enough for direct training. We randomly generate a parity-check matrix $\mathbf{H}$, train the weights with minibatch size $K = 50$, and evaluate the DER over M independently generated received words, where M is chosen large enough to give reliable estimates.

Fig. 3 shows the DER as a function of the number of training iterations for two randomly generated parity-check matrices. All three modifications yield a lower DER than the original DUD. In particular, the DER of HOM+ACC nearly converges within 2000 training iterations, whereas the original DUD has not yet converged at 20000 iterations.

To evaluate the transferability of HOM, we train the weights with the HOM+ACC scheme on one parity-check matrix and then evaluate the decoding performance for two other randomly generated parity-check matrices of the same parameters. Fig. 4 shows the results. The DER remains low even when the code being decoded differs from that used for training. This confirms that the homogenised weights transfer well to other parity-check matrices.

5 Zero-shot transfer to large QC-MDPC codes

Practical QC-MDPC codes have much longer code lengths than those considered in Section 4. For example, a proposed parameter set for 80-bit security [13] is

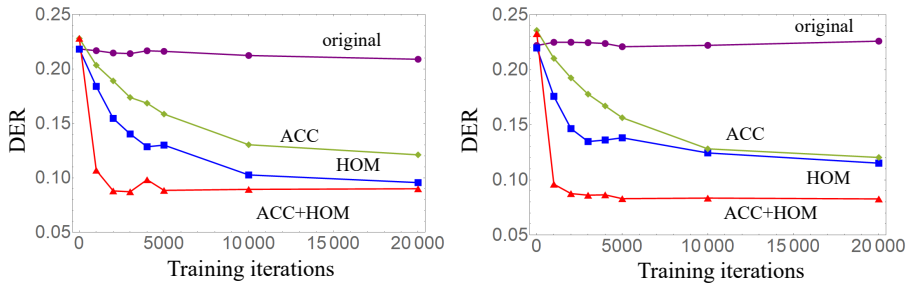


Fig. 3. The DER as a function of the number of training iterations for $(n, r) = (512, 256)$ and $w = 30$. Each panel shows four schemes: the original DUD, HOM, ACC, and HOM+ACC. The left and right panels correspond to two randomly generated parity-check matrices.

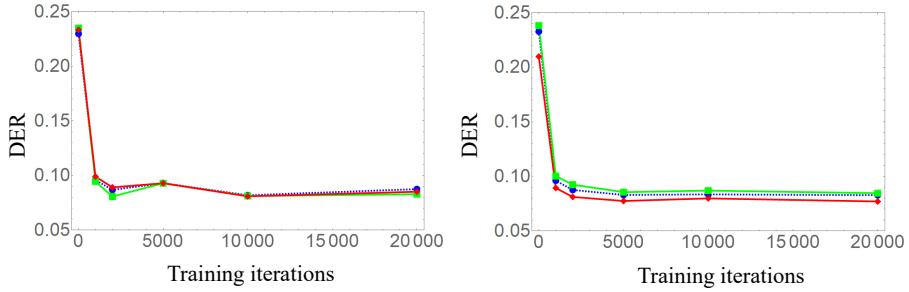


Fig. 4. The DER of HOM+ACC as a function of the number of training iterations ($n = 512$, $r = 256$, $w = 30$). In the left (right) panel, the parity-check matrix used for training is the same as in the left (right) panel of Fig. 3. The blue dashed lines show the DER when decoding the same code as used for training. For the red and green lines, we use the learned weights to decode codes defined by two other randomly generated parity-check matrices of the same parameters.

$(n, r, w, n_0) = (9602, 4801, 90, 2)$, giving $E = 432090$ edges in the Tanner graph. Without HOM, training would require optimising $L \cdot E$ weights, which is computationally infeasible. Even with HOM, which reduces the number of trainable parameters to L, the computational cost of each training step grows with the number of edges, making direct training on large codes impractical. As demonstrated in Section 4, however, the homogenised weights do not depend on the specific Tanner graph. We can therefore train the weights on a small code and apply them directly to larger codes without retraining—a zero-shot transfer.

To further reduce the DER, we employ two post-processing steps after Algorithm 2. The first is reproducing sum-product (RSP) [25]. When Algorithm 2 fails to find a valid codeword, RSP treats the current estimate $\vec{z}$ as a new received word and runs Algorithm 2 again. This process is repeated up to N_{RSP} times or until a valid codeword is found.

The second post-processing step, repeated correction (RC), is invoked when RSP fails. We sort the bit indices by ascending $|\Gamma_j|$, where $\vec{\Gamma}$ is the accumulated

LLR obtained from the first run of Algorithm 2, and select the B positions with the smallest values; these correspond to the bits for which the decoder has the least confidence. For each of the $2^B - 1$ non-empty subsets of these B positions, we flip the corresponding bits of the original received word $\vec{y}$ and run Algorithm 2 on the modified received word. The procedure terminates as soon as a valid codeword is found, or returns $\perp$ if all subsets have been exhausted. Algorithm 3 summarises this procedure. Note that RC uses the original received word $\vec{y}$ rather than the output of the failed RSP; in our experiments, this yields better results. In summary, decoding proceeds in three stages: Algorithm 2 with the learned weights, followed by RSP if unsuccessful, and then RC if RSP also fails.

Algorithm 3 Repeated correction (RC)

Input: original received word $\vec{y}$, accumulated LLR $\vec{\Gamma}$, number of candidate bits B, parity-check matrix $\mathbf{H}$.
Output: Estimated codeword $\vec{z}$, or decoding failure $\perp$.

Sort the bit indices so that $|\Gamma_{i_1}| \le |\Gamma_{i_2}| \le \cdots \le |\Gamma_{i_n}|$.
for each non-empty subset $\mathcal{S} \subseteq \{i_1, \ldots, i_B\}$ **do**
  $\vec{y}' \leftarrow \vec{y}$.
  for each $j \in \mathcal{S}$ **do**
    $y'_j \leftarrow y'_j \oplus 1$.
  end for
  $\vec{z} \leftarrow$ Algorithm 2($\vec{y}'$, mode = hard).
  if $\vec{z} \neq \perp$ **then**
    return $\vec{z}$.
  end if
end for
return $\perp$.

We evaluate the proposed decoder on two parameter sets for QC-MDPC-based cryptosystems. For 80-bit security, we use $(n, r, w, n_0) = (9602, 4801, 90, 2)$ [13], and for 128-bit security, $(n, r, w, n_0) = (19714, 9857, 142, 2)$. In both cases, we train the weights on a small code with $(n, r) = (512, 256)$ and $w = 30$, employing the HOM+ACC scheme with $L = 10$ BP iterations, minibatch size $K = 50$, and $L_{\text{learn}} = 2000$ training steps. The post-processing parameters are $N_{\text{RSP}} = 10$ and $B = 4$.

Fig. 5 shows the DER as a function of the error weight for the 80-bit security parameter set. We compare five methods: bit-flipping (BF), standard BP, BP with deep unfolding (BP+DL), BP+DL with RSP, and BP+DL with both RSP and RC. Each column uses a different set of weights learned in Section 4. Each panel uses a different randomly generated large parity-check matrix for evaluation. The similarity in performance across all four panels suggests that the DER is insensitive to the choice of training weights and the parity-check matrix for evaluation. In the top-left panel, we also plot BP+RSP, i.e., BP with reproducing sum-product but without deep unfolding. BP+DL+RSP achieves a substantially lower DER than BP+RSP across all error weights tested. Across all panels, BP+DL achieves a DER several orders of magnitude lower than standard BP, and each successive post-processing stage yields further improvement. Fig. 6 shows the corresponding results for the 128-bit security parameter set. The code used for evaluation is much larger than in the 80-bit case, while the

training code is the same. The learned weights still yield a DER well below that of standard BP.

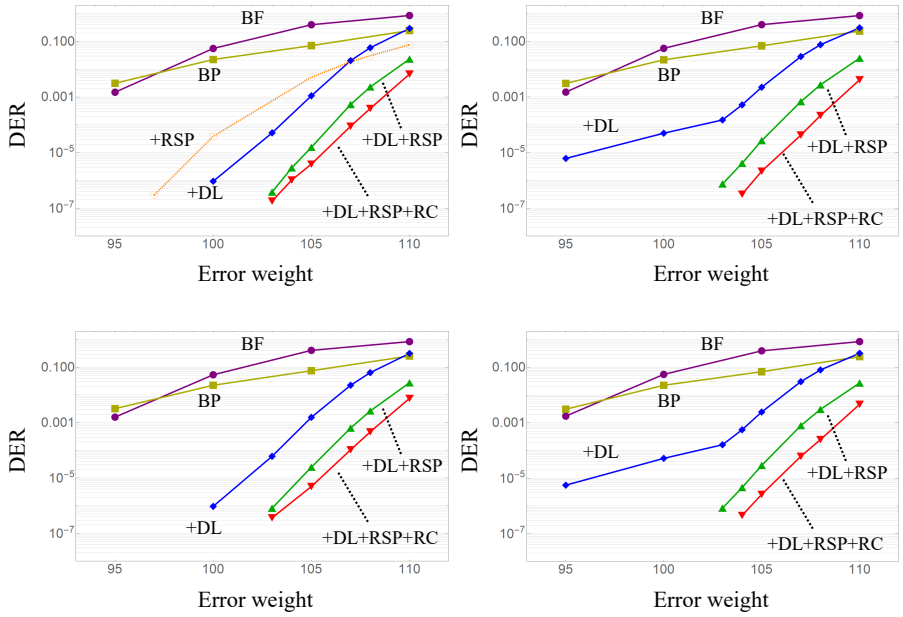


Fig. 5. The DER as a function of the error weight for $(n, r, w) = (9602, 4801, 90)$. Each column uses a different set of weights learned in Section 4. Each panel uses a different randomly generated large parity-check matrix for evaluation. Labels prefixed with "+" denote additions to BP. BP+RSP (orange, dashed) is shown in the top-left panel only.

6 Discussion and Summary

In this paper, we applied deep unfolding to BP decoding of QC-MDPC codes used in code-based cryptosystems. Deep unfolding interprets each iteration of BP as a layer of a neural network and introduces trainable weights that are optimised via training data. For the large QC-MDPC codes used in practical cryptosystems, however, direct training is impractical owing to the large number of edges in the Tanner graph. To overcome this difficulty, we proposed weight homogenisation, which constrains the trainable weights to a single scalar per iteration, making them independent of the specific Tanner graph. This enables a decoder trained on a small QC-MDPC code to be applied directly to larger codes. We also proposed LLR accumulation, which accelerates the convergence of training. Combining these two modifications, we trained the weights on a small code and demonstrated that they yield a lower DER than standard BP when applied to large QC-MDPC codes. Two post-processing steps, RSP and RC, further reduce the DER.

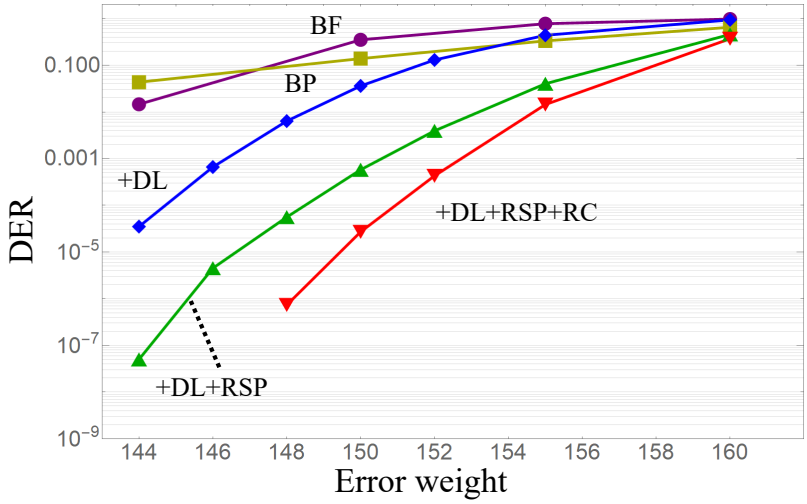


Fig. 6. The DER as a function of the error weight for $(n, r, w) = (19714, 9857, 142)$. The weights are learned on a small code with $(n, r) = (512, 256)$ and $w = 30$ in Section 4, and applied to a randomly generated large parity-check matrix for evaluation. Labels prefixed with "+" denote additions to BP.

Several directions for future work remain. First, the hyperparameters in this study, such as the number of BP iterations L, the minibatch size K, and the number of training steps L_{learn}, were not exhaustively optimised; a systematic search may yield further improvements. Second, our experiments evaluated the DER for a limited range of error weights. The error weights at which the DER meets the level required for cryptographic applications (e.g., 2^{-128} for 128-bit IND-CCA security) have not yet been determined; identifying these thresholds is an important next step. Third, in our experiments, the weights were trained on a single fixed parity-check matrix. Including data generated from multiple randomly chosen parity-check matrices within each minibatch may produce weights that generalise more effectively. Finally, RSP-ADMM [25] is reported to achieve a lower DER than our decoder for the 128-bit security parameter set; a detailed comparison, including decoding speed, is left for future work.

A BP decoding: derivation and vectorisation

This appendix presents the derivation of the BP algorithm reviewed in Section 3 and its vectorisation. We use the notation and channel model established in Sections 2 and 3: a codeword $\vec{x} \in C$ is transmitted over a memoryless BSC, producing a received word $\vec{y}$. The BP algorithm computes the marginal posterior probability $P(x_j \mid \vec{y})$ for each $j \in [n]$.

By Bayes' rule, the marginal posterior probability is

$$P(x_j \mid \vec{y}) = \sum_{\sim x_j} P(\vec{x} \mid \vec{y}) = \sum_{\sim x_j} \frac{P(\vec{x})\, P(\vec{y} \mid \vec{x})}{P(\vec{y})}, \tag{12}$$

where $\sum_{\sim x_j}$ denotes summation over all components of $\vec{x}$ except x_j. We assume that the transmitted word is chosen uniformly at random from C, which gives

$$P(\vec{x}) = \frac{1}{|C|}\,\mathbb{I}(\vec{x} \in C), \tag{13}$$

where $\mathbb{I}(\cdot)$ is the indicator function. The condition $\vec{x} \in C$ is equivalent to satisfying all parity-check equations, giving

$$\mathbb{I}(\vec{x} \in C) = \prod_{i \in [r]} \mathbb{I}\left(\sum_{\ell \in A_i} x_\ell = 0\right). \tag{14}$$

The memoryless assumption on the BSC gives

$$P(\vec{y} \mid \vec{x}) = \prod_{j \in [n]} P(y_j \mid x_j). \tag{15}$$

Combining these, we obtain

$$P(x_j \mid \vec{y}) = \frac{1}{|C|\,P(\vec{y})} \sum_{\sim x_j} \left(\prod_{j' \in [n]} P(y_{j'} \mid x_{j'})\right) \prod_{i \in [r]} \mathbb{I}\left(\sum_{\ell \in A_i} x_\ell = 0\right). \tag{16}$$

If $P(x_j = 0 \mid \vec{y}) > P(x_j = 1 \mid \vec{y})$, we set $\hat{x}_j = 0$; otherwise $\hat{x}_j = 1$. This rule is called symbol-wise maximum a posteriori (MAP) decoding. Evaluating the above expression directly requires summing over exponentially many terms. The BP algorithm computes these probabilities efficiently when the Tanner graph has no cycles.

BP, also known as sum-product message passing, is illustrated below using the parity-check matrix of Fig. 1, whose Tanner graph is a tree. The marginal posterior probability of x_3 expands as

$$\begin{aligned} P(x_3 \mid \vec{y}) \propto P(y_3 \mid x_3) \sum_{x_1} \mathbb{I}(x_1 + x_3 = 0)\, P(y_1 \mid x_1) \\ \times \sum_{x_4, x_6, x_7} \mathbb{I}(x_3 + x_4 + x_6 + x_7 = 0)\, P(y_4 \mid x_4)\, P(y_6 \mid x_6)\, P(y_7 \mid x_7) \\ \times \sum_{x_2, x_5} \mathbb{I}(x_2 + x_5 + x_4 = 0)\, P(y_2 \mid x_2)\, P(y_5 \mid x_5). \end{aligned} \tag{17}$$

This expression can be evaluated efficiently by working from the innermost sum outwards: first the sum over x_2 and x_5, then the sum over x_4, x_6, and x_7, and finally the sum over x_1. This avoids expanding the full 2^{n-1}-term sum.

The BP algorithm computes $P(x_3 \mid \vec{y})$ by passing messages on the tree rooted at v_3, as shown in Fig. 7(a). We define two messages on each edge: $M_{v_j \to c_i}(x_j)$, passed from the variable node to the check node, and $M_{c_i \to v_j}(x_j)$, passed from the check node to the variable node. These messages satisfy

$$M_{c_i \to v_j}(x_j) = \sum_{x_{A_i \setminus \{j\}}} \mathbb{I}\left(\sum_{\ell \in A_i \setminus \{j\}} x_\ell = x_j\right) \prod_{\ell \in A_i \setminus \{j\}} M_{v_\ell \to c_i}(x_\ell), \tag{18}$$

$$M_{v_j \to c_i}(x_j) = P(y_j \mid x_j) \prod_{\ell \in B_j \setminus \{i\}} M_{c_\ell \to v_j}(x_j), \tag{19}$$

where the outer sum in Eq. (18) ranges over all $x_{A_i\setminus\{j\}} = (x_\ell)_{\ell\in A_i\setminus\{j\}} \in \mathbb{F}_2^{|A_i|-1}$. These messages are evaluated from the leaves of the tree inwards, without any indeterminacy. The nodes v_2 and v_5 are each adjacent to c_2 only; the variable-to-check messages therefore reduce to

$$M_{v_2\to c_2}(x_2) = P(y_2 \mid x_2), \quad M_{v_5\to c_2}(x_5) = P(y_5 \mid x_5), \tag{20}$$

where the product in Eq. (19) is empty. Using these, the check-to-variable message $M_{c_2\to v_4}(x_4)$ is computed via Eq. (18) as

$$\begin{aligned} M_{c_2\to v_4}(x_4) &= \sum_{x_2,\,x_5} \mathbb{I}(x_2 + x_5 = x_4)\, M_{v_2\to c_2}(x_2)\, M_{v_5\to c_2}(x_5) \\ &= \begin{cases} M_{v_2\to c_2}(0)\, M_{v_5\to c_2}(0) + M_{v_2\to c_2}(1)\, M_{v_5\to c_2}(1), & x_4 = 0, \\ M_{v_2\to c_2}(0)\, M_{v_5\to c_2}(1) + M_{v_2\to c_2}(1)\, M_{v_5\to c_2}(0), & x_4 = 1, \end{cases} \end{aligned} \tag{21}$$

as illustrated in Fig. 7(b). Repeating this process along the tree, we obtain

$$P(x_3 \mid \vec{y}) \propto P(y_3 \mid x_3)\, M_{c_1\to v_3}(x_3)\, M_{c_3\to v_3}(x_3), \tag{22}$$

which corresponds to evaluating Eq. (17) from the innermost sum outwards. By choosing a different variable node as the root, the same algorithm computes $P(x_j \mid \vec{y})$ for any $j \in [n]$.

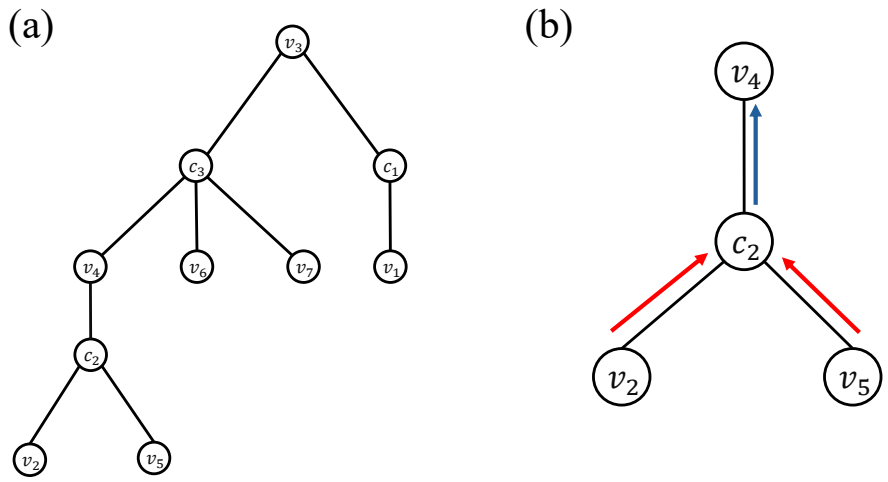


Fig. 7. (a) The Tanner graph of Fig. 1, redrawn as a tree rooted at variable node v_3. Messages are passed from the leaves towards the root. (b) The message-passing computation at check node c_2.

The non-iterative algorithm above cannot be applied directly when the Tanner graph contains cycles, because some messages remain undetermined. Moreover, computing $P(x_j \mid \vec{y})$ for every $j \in [n]$ by choosing each variable node as

the root in turn involves redundant computation; the same messages appear in the calculations for different roots. To address these issues, an iterative algorithm that updates all messages simultaneously is used instead. The iterative BP algorithm proceeds as follows:

(i) Initialise $M_{c_i \to v_j}(x_j) = 1$ for all $i \in [r]$, $j \in A_i$, and $x_j \in \{0, 1\}$.
(ii) Compute Eq. (19) for all $j \in [n]$ and $i \in B_j$.
(iii) Compute Eq. (18) for all $i \in [r]$ and $j \in A_i$.
(iv) For each $j \in [n]$, evaluate

$$g(x_j) = P(y_j \mid x_j) \prod_{i \in B_j} M_{c_i \to v_j}(x_j). \tag{23}$$

(v) Set

$$z_j = \begin{cases} 0, & g(0) \geq g(1), \\ 1, & \text{otherwise}, \end{cases} \tag{24}$$

for each $j \in [n]$.
(vi) If $\vec{z} = (z_1, \ldots, z_n)^T$ satisfies $\mathbf{H}\vec{z} = \vec{0}$, terminate. Otherwise, repeat from step (ii). The algorithm terminates after at most L iterations, where L is a prescribed upper bound.

The initialisation in step (i) allows this iterative algorithm to be formally applied to graphs with cycles. When the Tanner graph is cycle-free, the iterative algorithm converges to the same result as the non-iterative computation above. For graphs with few cycles, the algorithm is empirically known to perform well, although it no longer yields the exact marginal posterior probabilities. Performance degrades as the number of cycles increases.

Each message $M_{c_i \to v_j}(x_j)$ is a pair of values, $\{M_{c_i \to v_j}(0), M_{c_i \to v_j}(1)\}$. Only their ratio, however, is relevant for computing the marginal posterior probabilities, and hence each message is determined by a single scalar. It is convenient to represent the messages by their log ratios:

$$\alpha_{c_i \to v_j} = \ln \frac{M_{c_i \to v_j}(0)}{M_{c_i \to v_j}(1)}, \quad i \in [r],\ j \in A_i, \tag{25}$$

$$\beta_{v_j \to c_i} = \ln \frac{M_{v_j \to c_i}(0)}{M_{v_j \to c_i}(1)}, \quad j \in [n],\ i \in B_j. \tag{26}$$

We also define

$$\lambda_j = \ln \frac{P(y_j \mid 0)}{P(y_j \mid 1)}, \quad j \in [n]. \tag{27}$$

Using these variables, the iterative BP algorithm of steps (i)–(vi) is rewritten as follows:

(i) Initialise $\alpha_{c_i \to v_j} = 0$ for all $i \in [r]$ and $j \in A_i$.
(ii) For all $j \in [n]$ and $i \in B_j$, compute

$$\beta_{v_j \to c_i} = \lambda_j + \sum_{\ell \in B_j \setminus \{i\}} \alpha_{c_\ell \to v_j}. \tag{28}$$

(iii) For all $i \in [r]$ and $j \in A_i$, compute

$$\alpha_{c_i \to v_j} = 2 \tanh^{-1}\left(\prod_{\ell \in A_i \setminus \{j\}} \tanh \frac{\beta_{v_\ell \to c_i}}{2} \right). \tag{29}$$

(iv) For each $j \in [n]$, evaluate

$$\gamma_j = \lambda_j + \sum_{\ell \in B_j} \alpha_{c_\ell \to v_j}. \tag{30}$$

(v) Set

$$z_j = \begin{cases} 0, & \gamma_j \geq 0, \\ 1, & \text{otherwise}, \end{cases} \tag{31}$$

for each $j \in [n]$.

(vi) If $\vec{z} = (z_1, \ldots, z_n)^T$ satisfies $\mathbf{H}\vec{z} = \vec{0}$, terminate. Otherwise, repeat from step (ii). The algorithm terminates after at most L iterations.

To apply deep learning methods to the BP algorithm, it is convenient to vectorise the messages. The messages $\alpha_{c_i \to v_j}$ and $\beta_{v_j \to c_i}$ are each defined on an edge of the Tanner graph. Using the edge labels $e \in [E]$ and the incidence functions $\psi_v(e), \psi_c(e)$ introduced in Section 3, we define the inverse map $\phi(i, j) = e$ for each pair (i, j) with $(\mathbf{H})_{ij} = 1$; that is, $\psi_c(\phi(i, j)) = i$ and $\psi_v(\phi(i, j)) = j$. We collect the messages into E-dimensional vectors:

$$\vec{\alpha} = (\alpha_1, \ldots, \alpha_E)^T, \quad \alpha_e := \alpha_{c_{\psi_c(e)} \to v_{\psi_v(e)}}, \quad e \in [E], \tag{32}$$

$$\vec{\beta} = (\beta_1, \ldots, \beta_E)^T, \quad \beta_e := \beta_{v_{\psi_v(e)} \to c_{\psi_c(e)}}, \quad e \in [E]. \tag{33}$$

We also write $\vec{\lambda} = (\lambda_1, \ldots, \lambda_n)^T$ and $\vec{\gamma} = (\gamma_1, \ldots, \gamma_n)^T$.

We now vectorise the update rules, beginning with $\vec{\beta}$. For $e = \phi(i, j)$, Eq. (28) becomes

$$\begin{aligned} \beta_e &= \lambda_{\psi_v(e)} + \sum_{\ell \in B_j \setminus \{i\}} \alpha_{\phi(\ell, j)} \\ &= \lambda_{\psi_v(e)} + \sum_{\ell \in B_j} \alpha_{\phi(\ell, j)} - \alpha_e \\ &= \lambda_{\psi_v(e)} + \sum_{\substack{e' \in [E] \\ \psi_v(e') = \psi_v(e)}} \alpha_{e'} - \alpha_e, \end{aligned} \tag{34}$$

where the second line absorbs the excluded term $\alpha_e = \alpha_{\phi(i,j)}$ into the sum and subtracts it, and the third line replaces the summation over check nodes $\ell \in B_j$ by an equivalent summation over edges e' sharing the same variable node as e (Fig. 8). To express Eq. (34) in matrix form, we use the Kronecker delta and the connection matrix $\mathbf{U} \in \mathbb{F}_2^{n \times E}$ with $(\mathbf{U})_{j,e} = \delta_{j, \psi_v(e)}$. The first term satisfies $\lambda_{\psi_v(e)} = (\mathbf{U}^T \vec{\lambda})_e$ by definition of $\mathbf{U}$. For the summation over e', we decompose

the condition $\psi_v(e') = \psi_v(e)$ as

$$\sum_{\substack{e' \in [E] \\ \psi_v(e') = \psi_v(e)}} \alpha_{e'} = \sum_{e' \in [E]} \delta_{\psi_v(e),\, \psi_v(e')}\, \alpha_{e'} = \sum_{j' \in [n]} \delta_{\psi_v(e),\, j'} \sum_{e' \in [E]} \delta_{j',\, \psi_v(e')}\, \alpha_{e'} = (\mathbf{U}^T \mathbf{U}\, \vec{\alpha})_e, \tag{35}$$

where the second equality follows from the identity

$$\delta_{\psi_v(e),\, \psi_v(e')} = \sum_{j' \in [n]} \delta_{\psi_v(e),\, j'}\, \delta_{j',\, \psi_v(e')}. \tag{36}$$

Substituting into Eq. (34), we obtain

$$\vec{\beta} = \mathbf{U}^T \vec{\lambda} + (\mathbf{U}^T \mathbf{U} - \mathbf{I})\, \vec{\alpha}, \tag{37}$$

where $\mathbf{I}$ is the $E \times E$ identity matrix. Similarly, Eqs. (30) and (31) are vectorised as

$$\vec{\gamma} = \vec{\lambda} + \mathbf{U}\, \vec{\alpha}, \tag{38}$$

$$\vec{z} = \frac{\vec{1} + \operatorname{sign} \vec{\gamma}}{2}, \tag{39}$$

where sign acts elementwise and $\vec{1}$ denotes the all-one vector of dimension n.

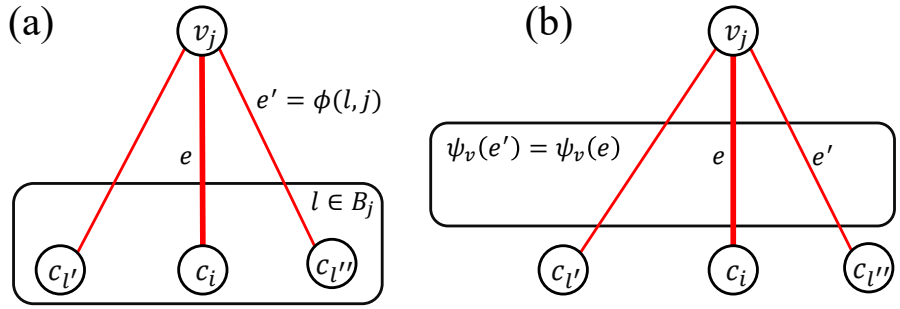


Fig. 8. The two equivalent summation ranges in Eq. (34). (a) The range $\ell \in B_j$: edges $e' = \phi(\ell, j)$ to the check nodes adjacent to v_j are shown in red. (b) The range $\psi_v(e') = \psi_v(e)$: edges sharing the same variable node as e are shown in red.

We vectorise Eq. (29) by treating the absolute value and the sign of α_e separately. Let $e = \phi(i, j)$. For the absolute value, Eq. (29) gives

$$|\alpha_e| = 2 \tanh^{-1}\left(\prod_{\ell \in A_i \setminus \{j\}} \left| \tanh \frac{\beta_{\phi(i,\ell)}}{2} \right| \right). \tag{40}$$

The product over ℓ does not directly yield a matrix–vector form. To convert it to a sum, we apply ln inside and compensate with exp:

$$\begin{aligned}|\alpha_e| &= 2\tanh^{-1}\left(\exp\left(\sum_{\ell\in A_i\setminus\{j\}} \ln\left|\tanh\frac{\beta_{\phi(i,\ell)}}{2}\right|\right)\right)\\ &= 2\tanh^{-1}\left(\exp\left(\sum_{\substack{e'\in[E]\\ \psi_c(e')=\psi_c(e)}} \ln\left|\tanh\frac{\beta_{e'}}{2}\right| - \ln\left|\tanh\frac{\beta_e}{2}\right|\right)\right),\end{aligned} \tag{41}$$

where the second equality rewrites the summation in terms of edges sharing the same check node as e, in the same manner as the derivation of Eq. (34). Let $\vec{\alpha}_{\mathrm{abs}} = (|\alpha_1|,\ldots,|\alpha_E|)^T$ and let $\mathbf{V}\in\mathbb{F}_2^{r\times E}$ denote the connection matrix with $(\mathbf{V})_{i,e} = \delta_{i,\psi_c(e)}$. Then

$$\vec{\alpha}_{\mathrm{abs}} = 2\tanh^{-1}\left(\exp\left((\mathbf{V}^T\mathbf{V}-\mathbf{I})\,\ln\left|\tanh\frac{\vec{\beta}}{2}\right|\right)\right), \tag{42}$$

where tanh, ln, exp, and the absolute value act elementwise on vectors; the same convention applies to all subsequent expressions.

For the sign, Eq. (29) gives

$$\operatorname{sign}\alpha_e = \prod_{\ell\in A_i\setminus\{j\}} \operatorname{sign}\beta_{\phi(i,\ell)} = \operatorname{sign}\beta_e \prod_{\substack{e'\in[E]\\ \psi_c(e')=\psi_c(e)}} \operatorname{sign}\beta_{e'}, \tag{43}$$

where the second equality follows from the same rewriting of the summation range as in Eqs. (34) and (41). To convert this product to a sum, we introduce the bijection

$$f\colon\{+1,-1\}\to\mathbb{F}_2,\quad f(x)=\frac{1-x}{2}, \tag{44}$$

which satisfies $f(xy) = f(x)+f(y) \pmod 2$. Applying f to Eq. (43), we obtain

$$f(\operatorname{sign}\alpha_e) \equiv f(\operatorname{sign}\beta_e) + \sum_{\substack{e'\in[E]\\ \psi_c(e')=\psi_c(e)}} f(\operatorname{sign}\beta_{e'}) \pmod 2. \tag{45}$$

Applying $f^{-1}(y) = 1-2y$ and substituting $f(\operatorname{sign}\beta_{e'}) = (1-\operatorname{sign}\beta_{e'})/2$, we obtain

$$\operatorname{sign}\alpha_e = \operatorname{sign}\beta_e \left(1 - 2\,\mathrm{mod}_2\left(\sum_{\substack{e'\in[E]\\ \psi_c(e')=\psi_c(e)}} \frac{1-\operatorname{sign}\beta_{e'}}{2}\right)\right), \tag{46}$$

where mod_2 denotes the reduction modulo 2 introduced in Algorithm 1. The same vectorisation procedure as in Eq. (41) gives

$$\operatorname{sign}\vec{\alpha} = \left(\vec{1} - 2\,\mathbf{V}^T\,\mathrm{mod}_2\left(\mathbf{V}\,\frac{\vec{1}-\operatorname{sign}\vec{\beta}}{2}\right)\right)\odot\operatorname{sign}\vec{\beta}. \tag{47}$$

Here, the direct vectorisation of Eq. (46) yields

$$\mathrm{mod}_2\left(\mathbf{V}^T\mathbf{V}\,\frac{\vec{1}-\mathrm{sign}\,\vec{\beta}}{2}\right). \tag{48}$$

Each row of $\mathbf{V}^T$, however, contains exactly one nonzero entry; $\mathbf{V}^T$ therefore merely copies a single component of its argument without addition. mod_2 may hence be applied after $\mathbf{V}$ and before $\mathbf{V}^T$, yielding Eq. (47). From Eqs. (42) and (47), $\vec{\alpha}$ is recovered as

$$\vec{\alpha} = \mathrm{sign}\,\vec{\alpha} \odot \vec{\alpha}_{\mathrm{abs}}. \tag{49}$$

Eqs. (37), (42), (47), (49), (38), and (39) together constitute Algorithm 1.

References

1. G. Alagic, M. Bros, P. Ciadoux, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, H. Silberg, D. Smith-Tone, and N. Waller. Status report on the fourth round of the NIST post-quantum cryptography standardization process. NIST Internal Report 8545, 2025.
2. N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. Aguilar Melchor, R. Misoczki, E. Persichetti, J. Richter-Brockmann, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor. BIKE: Bit flipping key encapsulation, specification document, version 2024.10.10.1, 2024. Available at `https://bikesuite.org/`.
3. E. Berlekamp, R. McEliece, and H. Van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.
4. D. J. Bernstein, J. Buchmann, and E. Dahmen, editors. *Post-Quantum Cryptography*. Springer, 2009.
5. J. Chaulet and N. Sendrier. Worst case QC-MDPC decoder for McEliece cryptosystem. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1366–1370. IEEE, 2016.
6. W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
7. R. G. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28, 1962.
8. Q. Guo, T. Johansson, and P. S. Wagner. A key recovery reaction attack on QC-MDPC. *IEEE Transactions on Information Theory*, 65(3):1845–1861, 2019.
9. J. R. Hershey, J. Le Roux, and F. Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*, 2014.
10. K. Kurosawa and K. Inaba. Consideration on the NP completeness of linear codes. *The Transactions of the Institute of Electronics, Information and Communication Engineers. A*, J68-A(9):953–956, 1985.
11. X. Liu and S. C. Draper. The ADMM penalized decoder for LDPC codes. *IEEE Transactions on Information Theory*, 62(6):2966–2984, 2016.
12. R. J. McEliece. A public key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 42–44:114–116, 1978.
13. R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. L. M. Barreto. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *2013 IEEE International Symposium on Information Theory*, pages 2069–2073. IEEE, 2013.

14. E. Nachmani, Y. Be'ery, and D. Burshtein. Learning to decode linear codes using deep learning. In *Proceedings of the 54th Annual Allerton Conference on Communication, Control, and Computing*, pages 341–346. IEEE, 2016.
15. E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery. Deep learning methods for improved decoding of linear codes. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):119–131, 2018.
16. National Institute of Standards and Technology. Post-Quantum Cryptography. `https://csrc.nist.gov/Projects/post-quantum-cryptography/`.
17. National Institute of Standards and Technology. Module-lattice-based digital signature standard. Federal Information Processing Standards Publication 204, 2024.
18. National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard. Federal Information Processing Standards Publication 203, 2024.
19. National Institute of Standards and Technology. Stateless hash-based digital signature standard. Federal Information Processing Standards Publication 205, 2024.
20. A. Nilsson, I. E. Bocharova, B. D. Kudryashov, and T. Johansson. A weighted bit flipping decoder for QC-MDPC-based cryptosystems. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 1266–1271. IEEE, 2021.
21. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
22. N. Sendrier and V. Vasseur. On the decoding failure rate of QC-MDPC bit-flipping decoders. In *Post-Quantum Cryptography (PQCrypto 2019)*, volume 11505 of *Lecture Notes in Computer Science*, pages 404–416. Springer, 2019.
23. P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994.
24. R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.
25. K. Watanabe, M. Ohtsuka, and Y. Tsukie. ADMM and reproducing sum-product decoding algorithm applied to QC-MDPC code-based McEliece cryptosystems. *IEEE Transactions on Information Theory*, 70(3):1774–1786, 2024.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.