PAPER *Special Section on Information Theory and Its Applications*

Complexity Reduction of the Gazelle and Snyders Decoding Algorithm for Maximum Likelihood Decoding

Hideki YAGI[†a)], ***Student Member***, **Manabu KOBAYASHI**[††], ***Regular Member***, ***and*** **Shigeichi HIRASAWA**[†], ***Fellow***

SUMMARY Several reliability based code search algorithms for maximum likelihood decoding have been proposed. These algorithms search the most likely codeword, using the most reliable information set where the leftmost k (the dimension of code) columns of generator matrix are the most reliable and linearly independent. Especially, D. Gazelle and J. Snyders have proposed an efficient decoding algorithm and this algorithm requires small number of candidate codewords to find out the most likely codeword. In this paper, we propose new efficient methods for both generating candidate codewords and computing metrics of candidate codewords to obtain the most likely codeword at the decoder. The candidate codewords constructed by the proposed method are identical those in the decoding algorithm of Gazelle et al. Consequently, the proposed decoding algorithm reduces the time complexity in total, compared to the decoding algorithm of Gazelle et al. without the degradation in error performance.

key words: *maximum likelihood decoding, information set decoding, most reliable basis, reliability measure, linear block codes*

1. Introduction

Soft decision decoding for linear block codes reduces the block error probability by taking advantage of channel measurement information, compared with conventional hard decision decoding. Particularly, maximum likelihood decoding (MLD) achieves the best error performance when each codeword has the equal probability to be transmitted. Since maximum likelihood (ML) decoder rapidly becomes too complex to implement as the code length becomes large, many researchers have been devoted to develop new decoding algorithms to reduce the time and space complexity of MLD.

There are, in general, two types of efficient MLD algorithms. The first type is trellis-based MLD algorithm such as the Viterbi algorithm [1] or recursive MLD (RMLD) algorithm [2]. Trellis-based MLD algorithms are "breadth-first" search algorithm [3] which reduces the maximum number of computations. For longer and medium to high rate codes, however, the space complexity, $O(2^k)$, is large where k represents the dimension of code. The second type of efficient MLD algorithms is "depth-first" search algorithm [3] which iteratively generate candidate codewords. The decoding algorithms of this type reduce the average number of computations and they are known to be efficient at moderate or high signal to noise ratio (SNR) with small space complexity. In this paper, we focus on the second type of MLD algorithms.

Some of the optimal and sub-optimal MLD algorithms of the second type are called the reliability-based ordered information set decoding algorithms which use *the most reliable basis* (MRB) [3]–[8]. The MRB based information set decoding algorithms reduce the time complexity as well as the space one.

D. Gazelle and J. Snyders have proposed a decoding algorithm which effectively generates candidate codewords for the ML codeword based on the MRB [4]. Hereafter, we will call this algorithm the GS decoding algorithm. This algorithm effectively eliminates unnecessary candidate codewords. Consequently, the GS decoding algorithm requires smaller number of candidate codewords than that of other MRB based MLD algorithms with small space complexity. At low SNR, for moderate code rates and large code lengths, however, the time complexity required for performing MLD is still impractically large.

In this paper, first we propose a new method for constructing candidate codewords by exploiting codewords constructed so far and the generating rule of candidate codewords. This method reduces the time complexity to construct a candidate codeword from $O(kn)$ to $O(n)$, where n represents the code length. Based on the same approach as the above method, we derive an effective method for computing metrics of candidate codewords. Next, we present the proposed decoding algorithm, which is an improved version of GS decoding algorithm, by employing these methods. Consequently, we show the proposed decoding algorithm reduces the time complexity of the GS decoding algorithm, while the proposed algorithm maintains the best error performance.

This paper is organized as follows. In Sect. 2, we describe the MRB based MLD algorithm as a preliminary. In Sect. 3, we briefly review the GS decoding algorithm. In Sect. 4, we propose a computationally efficient method for constructing candidate codewords

Manuscript received January 17, 2003.
Manuscript revised April 18, 2003.
Final manuscript received June 6, 2003.

†The authors are with the Department of Industrial and Management Systems Engineering, School of Science and Engineering, Waseda University, Tokyo, 169-8555 Japan.
††The author is with the Department of Information Science, School of Engineering, Shonan Institute of Technology, Fujisawa-shi, 251-8511 Japan.
a) E-mail: yagi@hirasa.mgmt.waseda.ac.jp

and present a new MLD algorithm. Finally, some simulation results are presented in Sect. 5 and concluding remarks are given in Sect. 6.

2. The MRB Based MLD Algorithm

Let $\mathcal{C}$ be a binary linear (n,k,d) block code of length n, dimension k and minimum distance d. Let G be the generator matrix of $\mathcal{C}$. Assume that each codeword $\boldsymbol{c}=(c_1,c_2,\cdots,c_n)\in\mathcal{C}$ has the equal probability to be transmitted through the Additive White Gaussian Noise (AWGN) channel of the signal to noise ratio E_b/N_0. The detector projects the received sequence $\boldsymbol{r}=(r_1,r_2,\cdots,r_n)\in\mathcal{R}^n$ into the reliability sequence $\boldsymbol{\theta}=(\theta_1,\theta_2,\cdots,\theta_n)$, where $\theta_j=\ln\frac{P(r_j|c_j=0)}{P(r_j|c_j=1)}$, $j=1,2,\cdots,n$, and inputs $\boldsymbol{\theta}$ into the decoder. The decoder estimates a transmitted codeword from both $\boldsymbol{\theta}$ and the hard decision received sequence $\boldsymbol{z}=(z_1,z_2,\cdots,z_n)\in\{0,1\}^n$ from $\boldsymbol{\theta}$ where

$$z_j=\begin{cases}0, & \text{if } \theta_j\geq 0;\\ 1, & \text{otherwise.}\end{cases} \tag{1}$$

An error probability of z_j, $P(z_j\neq c_j|r_j)$, is smaller as the value $|\theta_j|$, $j=1,2,\cdots,n$, becomes larger. Therefore, we call $|\theta_j|$ *reliability measure*.

For any n-tuple $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)\in\{0,1\}^n$, let $L(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\theta})$ be the function of *the reliability loss* with respect to $\boldsymbol{z}$, defined as

$$L(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\theta})=\sum_{j=1}^{n}(x_j\oplus z_j)|\theta_j|, \tag{2}$$

where $\oplus$ represents the exclusive OR operator. We will use $L(\boldsymbol{x})$ in place of $L(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\theta})$ for simplicity if we can fix $\boldsymbol{z}$ and $\boldsymbol{\theta}$ from the context. Then the ML codeword $\boldsymbol{c}_{ML}\in\mathcal{C}$ satisfies $L(\boldsymbol{c}_{ML})=\min_{\boldsymbol{c}\in\mathcal{C}}L(\boldsymbol{c})$ [10], [11].

At first, the MRB based decoder reorders columns of generator matrix G in the nonincreasing order of reliability measure. We denote the resultant generator matrix by $\bar{G}$. Let $\bar{\mathcal{C}}$ be the code generated by $\bar{G}$. Define that $\bar{\boldsymbol{\theta}}$ and $\bar{\boldsymbol{z}}$ are the ordered sequences of $\boldsymbol{\theta}$ and $\boldsymbol{z}$, respectively, in the same ordering of columns of $\bar{G}$, i.e., $|\bar{\theta}_{j_1}|\geq|\bar{\theta}_{j_2}|$, $1\leq j_1<j_2\leq n$. This reordering defines a permutation function λ_1 such as $\bar{\boldsymbol{\theta}}=\lambda_1(\boldsymbol{\theta})$.

Furthermore, columns of $\bar{G}$ are permuted so that the leftmost k columns are *the most reliable and linearly independent* (MRI). MRI columns are linearly independent k columns of generator matrix whose reliabilities are the largest among any other linearly independent k columns. For the resultant matrix, the leftmost $k\times k$ matrix is rearranged to be the identity matrix by the standard row operations. This identity matrix forms MRB. The resultant generator matrix with MRB is denoted by $\tilde{G}$. The bit positions of $\bar{\boldsymbol{\theta}}$ and $\bar{\boldsymbol{z}}$ are reordered to be $\tilde{\boldsymbol{\theta}}$ and $\tilde{\boldsymbol{z}}$, respectively, in the same reordering manner of columns of $\tilde{G}$. Note that $|\tilde{\theta}_j|\geq|\tilde{\theta}_{j+1}|$, $1\leq j\leq k-1$, and $|\tilde{\theta}_{j'}|\geq|\tilde{\theta}_{j'+1}|$, $k+1\leq j'\leq n-1$. This reordering defines a second permutation function λ_2 such that $\tilde{\boldsymbol{\theta}}=\lambda_2(\bar{\boldsymbol{\theta}})$. Let $\tilde{\mathcal{C}}$ denote the code generated by $\tilde{G}$, which is equivalent to $\mathcal{C}$.

Define that $\boldsymbol{u}=(u_1,u_2,\cdots,u_k)\in\{0,1\}^k$ consists of MRI symbols of $\tilde{\boldsymbol{z}}=(\tilde{z}_1,\tilde{z}_2,\cdots,\tilde{z}_n)$. i.e., $u_j=\tilde{z}_j$, $j=1,2,\cdots,k$. $\boldsymbol{u}$ is regarded as an information sequence and the decoder generates the initial candidate codeword $\tilde{\boldsymbol{c}}_0=\boldsymbol{u}\tilde{G}$. Remark that $\tilde{\boldsymbol{c}}_0$ is the ML codeword if $\tilde{\boldsymbol{c}}_0=\tilde{\boldsymbol{z}}$ [9], [10]. If $L(\tilde{\boldsymbol{c}}_0)>0^\dagger$, the decoder iteratively constructs candidate codewords by $\tilde{G}$ and searches the ML codeword which minimizes the reliability loss. The decoder outputs the ML codeword as the estimated codeword at the end of decoding procedure.

3. A Brief Review of the GS Decoding Algorithm

In this section, we review the GS decoding algorithm, which effectively generates candidate codewords [4].

Definition 1: Define $\kappa=\min\{k,n-k-1\}$ and let $w_H(\boldsymbol{x})$ denote the Hamming weight of a vector $\boldsymbol{x}$. For $l=1,2,\cdots,\kappa$, let $\mathcal{T}_l=\{\boldsymbol{t}\in\{0,1\}^k|w_H(\boldsymbol{t})=l\}$ be the set of *test error patterns* with the Hamming weight l. We call $\tilde{\boldsymbol{w}}=\boldsymbol{t}\tilde{G}$, $\boldsymbol{t}\in\mathcal{T}_l$, *test error codewords*. Then the candidate codeword $\tilde{\boldsymbol{c}}$ is denoted by $\tilde{\boldsymbol{c}}=\tilde{\boldsymbol{c}}_0\oplus\tilde{\boldsymbol{w}}$. □

The GS decoder processes $\mathcal{T}_l$ in increasing order of l. The processing of $\mathcal{T}_l$ is referred to as *phase-l reprocessing*. This terminology is given in [5], [6].

In phase-l reprocessing, $l=1,2,\cdots,\kappa$, a test error pattern $\boldsymbol{t}_i=(t_{i,1},t_{i,2},\cdots,t_{i,k})$, $i=1,2,\cdots,\binom{k}{l}$, is generated in the increasing order of standard binary representation, i.e., the inequality $\sum_{j=1}^{k}t_{p,j}2^{k-j}<\sum_{j=1}^{k}t_{q,j}2^{k-j}$ holds for arbitrary pairs $\boldsymbol{t}_p,\boldsymbol{t}_q\in\mathcal{T}_l$, $1\leq p<q\leq\binom{k}{l}$.

In phase-l reprocessing, a test error pattern $\boldsymbol{t}_i$, $i>1$, is generated from $\boldsymbol{t}_{i-1}$ in the following manner. First, let the topmost element of $\mathcal{T}_l$ be $\boldsymbol{t}_1=(0^{k-l},1^l)$ where $(0^\alpha,1^\beta)\in\{0,1\}^{\alpha+\beta}$ consists of α consecutive 0's and β consecutive 1's. For $i=2,3,\cdots,\binom{k}{l}$, we find the rightmost bit position J such that $(t_{i-1,J},t_{i-1,J+1})=(0,1)$, i.e., $J=\max\{j|\,(t_{i-1,j},t_{i-1,j+1})=(0,1)\}$. Then we set $(t_{i,1},t_{i,2},\cdots,t_{i,J-1})=(t_{i-1,1},t_{i-1,2},\cdots,t_{i-1,J-1})$ and replace the subsequence $(t_{i-1,J},t_{i-1,J+1})=(0,1)$ with $(t_{i,J},t_{i,J+1})=(1,0)$. Afterwards, $\boldsymbol{t}_i$ can be obtained by setting $(0^\alpha,1^\beta)$ at the next positions of $J+1$ where β is determined to satisfy $w_H(\boldsymbol{t}_i)=l$. That means we set $(t_{i,J+2},t_{i,J+3},\cdots,t_{i,k})=(0^\alpha,1^\beta)$. Hereafter we will call this method, which generates $\boldsymbol{t}_i$ from $\boldsymbol{t}_{i-1}$, *the generation rule A*.

---

†If we can fix $\tilde{\boldsymbol{x}}=\lambda_2(\lambda_1(\boldsymbol{x}))$ and $\tilde{\boldsymbol{\theta}}$ from the context, we will denote $L(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{z}},\tilde{\boldsymbol{\theta}})$ by $L(\tilde{\boldsymbol{x}})$ for simplicity. Remark that $L(\tilde{\boldsymbol{x}},\tilde{\boldsymbol{z}},\tilde{\boldsymbol{\theta}})=L(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\theta})$ from the property of AWGN.

During decoding procedure, the decoder iteratively constructs a test error codeword $\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i\tilde{G}$. The decoder searches the test error codeword $\tilde{\boldsymbol{w}}_{ML}$ which satisfies $\tilde{\boldsymbol{c}}_{ML} = \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}_{ML}$, where $\tilde{\boldsymbol{c}}_{ML}$ is the ML codeword.

We here define the set of codewords $\tilde{\mathcal{C}}_{l,s}$ such that $\tilde{\mathcal{C}}_{l,s}$ includes all generated candidate codewords $\tilde{\boldsymbol{c}}_i = \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}_i$ before generating a test error pattern $\boldsymbol{t}_s \in \mathcal{T}_l$, i.e.,

$$\begin{aligned}\tilde{\mathcal{C}}_{l,s} &= \{\tilde{\boldsymbol{c}}_0\}\\ &\cup \bigcup_{j=1}^{l-1} \left\{ \tilde{\boldsymbol{c}}_i = \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}_i \middle| \tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i\tilde{G}, \boldsymbol{t}_i \in \mathcal{T}_j, i = 1, 2, \cdots, \binom{k}{j} \right\}\\ &\cup \{\tilde{\boldsymbol{c}}_i = \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}_i | \tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i\tilde{G}, \boldsymbol{t}_i \in \mathcal{T}_l, i = 1, 2, \cdots, s-1\}. \quad (3)\end{aligned}$$

Let $\underline{L}_{l,s}$ be the minimum value of the reliability loss in $\tilde{\mathcal{C}}_{l,s}$, i.e., $\underline{L}_{l,s} = \min_{\tilde{\boldsymbol{c}} \in \tilde{\mathcal{C}}_{l,s}} L(\tilde{\boldsymbol{c}})$. For $\boldsymbol{t}_i$, let $\Delta(\boldsymbol{t}_i) = \sum_{j=1}^{k} t_{i,j}|\tilde{\theta}_j|$ denote the reliability loss with respect to $\boldsymbol{u}$. In phase-l reprocessing, $\boldsymbol{t}_i$ needs not to be encoded if $\boldsymbol{t}_i \in \mathcal{T}_l$ satisfies the following inequality[†]:

$$\underline{L}_{l,i} \leq \Delta(\boldsymbol{t}_i). \quad (4)$$

It is because a candidate codeword $\tilde{\boldsymbol{c}}_i = \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}_i$, $\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i\tilde{G}$, always satisfies $\Delta(\boldsymbol{t}_i) \leq L(\tilde{\boldsymbol{c}}_i)$.

The GS decoding algorithm effectively eliminates some test error patterns that successively satisfy Eq. (4). Let $\boldsymbol{t}_p$ be the last generated test error pattern before generating $\boldsymbol{t}_i$ and we assume that Eq. (4) holds for $\boldsymbol{t}_i$. Then the next test error pattern $\boldsymbol{t}_s$ is generated in the following manner. First, find the bit position I satisfying $(t_{p,I}, t_{p,I+1}) = (0, 1)$ and $(t_{i,I}, t_{i,I+1}) = (1, 0)$. We here consider the temporary test error pattern $\hat{\boldsymbol{t}} = (\hat{t}_1, \hat{t}_2, \cdots, \hat{t}_k)$ as

$$\hat{t}_j = \begin{cases} t_{i,j}, & \text{for } 1 \leq j \leq I-1; \\ 0, & \text{for } I \leq j \leq k. \end{cases} \quad (5)$$

If $w_H(\hat{\boldsymbol{t}}) = 0$, eliminate the rest of elements in $\mathcal{T}_l$. Otherwise $\boldsymbol{t}_s$ is obtained by replacing the rightmost subsequence $(0, 1)$ in $\hat{\boldsymbol{t}}$ with $(1, 0)$ and setting $(0^\alpha, 1^\beta)$ at the next positions of this subsequence $(1, 0)$, where β is determined to satisfy $w_H(\boldsymbol{t}_s) = l$ [4][††]. Hereafter we will call this method, which generates $\boldsymbol{t}_s$ from $\boldsymbol{t}_i$ and $\boldsymbol{t}_p$, *the generation rule B*.

Lemma 1 ([4]): Assume that $\boldsymbol{t}_i$ satisfies Eq. (4) and $\boldsymbol{t}_s$ is generated by the generation rule B. If there exists s', $i < s' < s$, then

$$\Delta(\boldsymbol{t}_{s'}) \geq \Delta(\boldsymbol{t}_i) \geq \underline{L}_{l,i} = \underline{L}_{l,s'}. \quad (6)$$

Since $\Delta(\boldsymbol{t}_{s'}) \geq \underline{L}_{l,s'}$, such $\boldsymbol{t}_{s'}$ needs not to be encoded. □

Example 1: In phase-4 reprocessing, let $\boldsymbol{t}_p = (0000110\cdots0110)$ and $\boldsymbol{t}_i = (0000110\cdots1001)$ and assume that $\boldsymbol{t}_i$ satisfies Eq. (4). Then the next test error pattern $\boldsymbol{t}_{i+1} = (0000110\cdots1010)$ also satisfies Eq. (4), since $\Delta(\boldsymbol{t}_i) \leq \Delta(\boldsymbol{t}_{i+1})$. According to the generation rule B, we find $I = k-3$ from $\boldsymbol{t}_p$ and $\boldsymbol{t}_i$ and we construct $\hat{\boldsymbol{t}} = (0000110\cdots0000)$. Since $w_H(\hat{\boldsymbol{t}}) = 2 \neq 0$, the rightmost subsequence $(\hat{t}_4, \hat{t}_5) = (0, 1)$ in $\hat{\boldsymbol{t}}$ is replaced with $(1, 0)$ and we obtain $(t_{s,1}, t_{s,2}, \cdots, t_{s,5}) = (00010)$. At the positions to the right of $j = 5$, we set $(t_{s,6}, t_{s,7}, \cdots, t_{s,k}) = (00\cdots0111)$ such that $w_H(\boldsymbol{t}_s) = 4$. The next test error pattern to be generated is $\boldsymbol{t}_s = (000100\cdots00111)$. □

Example 2: In phase-4 reprocessing, let $\boldsymbol{t}_p = (0000110\cdots1001)$ and $\boldsymbol{t}_i = (0001000\cdots0111)$ and assume that $\boldsymbol{t}_i$ satisfies Eq. (4). We find $I = 4$ from $\boldsymbol{t}_p$ and $\boldsymbol{t}_i$ and we construct $\hat{\boldsymbol{t}} = (00000\cdots0000)$. Since $w_H(\hat{\boldsymbol{t}}) = 0$, there remains no test error pattern $\boldsymbol{t}_s \in \mathcal{T}_4$ to be generated. Then we start phase-5 reprocessing and generate $\boldsymbol{t}_1 = (000\cdots011111) \in \mathcal{T}_5$. □

For a candidate codeword $\tilde{\boldsymbol{c}}_i = \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}_i$, a function $\Lambda(\cdot)$ with respect to $\tilde{\boldsymbol{c}}_0$ is defined as

$$\Lambda(\tilde{\boldsymbol{w}}_i) = \sum_{j=1}^{n} (-1)^{\tilde{z}_j \oplus \tilde{c}_{0,j}} \tilde{w}_{i,j} |\tilde{\theta}_j|. \quad (7)$$

Then, we can obtain $L(\tilde{\boldsymbol{c}}_i)$ such that

$$L(\tilde{\boldsymbol{c}}_i) = L(\tilde{\boldsymbol{c}}_0) + \Lambda(\tilde{\boldsymbol{w}}_i), \quad (8)$$

since

$$\begin{aligned} L(\tilde{\boldsymbol{c}}_i) &= \sum_{j=1}^{n} (\tilde{z}_j \oplus \tilde{c}_{0,j} \oplus \tilde{w}_{i,j})|\tilde{\theta}_j| \\ &= \sum_{j=1}^{n} (\tilde{z}_j \oplus \tilde{c}_{0,j})|\tilde{\theta}_j| + \sum_{j=1}^{n} (-1)^{\tilde{z}_j \oplus \tilde{c}_{0,j}} \tilde{w}_{i,j}|\tilde{\theta}_j|. \end{aligned} \quad (9)$$

At some decoding stage, we assume that $\tilde{\boldsymbol{w}}^*$ minimizes Eq. (7) in $\tilde{\mathcal{C}}_{l,s}$. Let $\underline{\Lambda}$ denote the minimum value, i.e., $\underline{\Lambda} = \Lambda(\tilde{\boldsymbol{w}}^*) \leq 0$. Thereafter, the decoder searches a test error codeword $\tilde{\boldsymbol{w}}_j \in \tilde{\mathcal{C}} \backslash \tilde{\mathcal{C}}_{l,s}$ such that $\Lambda(\tilde{\boldsymbol{w}}_j) < \underline{\Lambda}$ and updates $\underline{\Lambda}$ as $\underline{\Lambda} := \Lambda(\tilde{\boldsymbol{w}}_j)$.

We now describe the GS decoding algorithm.

[The GS decoding algorithm]

1) Generate $\tilde{\boldsymbol{c}}_0 := \boldsymbol{u}\tilde{G}$, and set $\underline{L} := L(\tilde{\boldsymbol{c}}_0)$, $\tilde{\boldsymbol{w}}^* := \boldsymbol{0}$, $\underline{\Lambda} := 0$ and $l := 1$.
2) a) Generate $\boldsymbol{t}_1 \in \mathcal{T}_l$ and calculate $\Delta(\boldsymbol{t}_1)$. If $\underline{L} < \Delta(\boldsymbol{t}_1)$, then output $\tilde{\boldsymbol{c}}_{ML} := \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}^*$ and stop. Otherwise generate $\tilde{\boldsymbol{w}}_1 := \boldsymbol{t}_1\tilde{G}$.

[†]Though Gazelle et al. have presented less stringent sufficient conditions than Eq. (4) in [4], we will not describe them for simplicity. However, all sufficient conditions presented in [4] can be also applicable to the proposed decoding algorithms in this paper.

[††]If $w_H(\hat{\boldsymbol{t}}) \neq 0$ and $\hat{\boldsymbol{t}}$ has no subsequence $(0, 1)$, then eliminate the rest of elements in $\mathcal{T}_l$.

b) Calculate $\Lambda(\tilde{\boldsymbol{w}}_1)$. If $\Lambda(\tilde{\boldsymbol{w}}_1) < \underline{\Lambda}$, then $\underline{\Lambda} := \Lambda(\tilde{\boldsymbol{w}}_1)$, $\underline{L} := L(\tilde{\boldsymbol{c}}_0) + \underline{\Lambda}$ and $\tilde{\boldsymbol{w}}^* := \tilde{\boldsymbol{w}}_1$.

3) Set $p := 1$, $i := 2$.

a) Generate $\boldsymbol{t}_i$ from $\boldsymbol{t}_p$ by the generation rule A.

b) Calculate $\Delta(\boldsymbol{t}_i)$. If $\underline{L} < \Delta(\boldsymbol{t}_i)$, then try to generate the next error pattern $\boldsymbol{t}_s$ by the generation rule B, otherwise go to 3-c). If there exists $\boldsymbol{t}_s$, then set $\boldsymbol{t}_p := \boldsymbol{t}_i$, $\boldsymbol{t}_i := \boldsymbol{t}_s$ and go to 3-b), otherwise go to 4).

c) Set $\tilde{\boldsymbol{w}}_i := \boldsymbol{t}_i\tilde{G}$ and calculate $\Lambda(\tilde{\boldsymbol{w}}_i)$. If $\Lambda(\tilde{\boldsymbol{w}}_i) < \underline{\Lambda}$, then $\underline{\Lambda} := \Lambda(\tilde{\boldsymbol{w}}_i)$, $\underline{L} := L(\tilde{\boldsymbol{c}}_0) + \underline{\Lambda}$ and $\tilde{\boldsymbol{w}}^* := \tilde{\boldsymbol{w}}_i$. Set $\boldsymbol{t}_p := \boldsymbol{t}_i$, $i := i + 1$ and go to 3-a).

4) Set $l := l + 1$. If $l \leq \kappa$, then go to 2), otherwise output $\tilde{\boldsymbol{c}}_{ML} := \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}^*$ and stop. □

In the algorithm, for each l and i, the minimum value of reliability loss $\underline{L}_{l,i}$ is obtained and we update $\underline{L}$ such that $\underline{L} := \underline{L}_{l,i}$.

Note that the GS decoding algorithm is a structured MLD version of information set decoding [12].

We here discuss the complexity of the GS decoding algorithm. The time complexity of permuting $\boldsymbol{\theta}$ in the nonincreasing order is $O(n \log n)$ and the construction of $\tilde{G}$ requires $O(n \times \kappa^2)$ [3]–[5]. These procedures are carried out only once in a decoding procedure. Contrary to these procedures, encoding test error patterns $\boldsymbol{t}_i$ are carried out iteratively, where each encoding requires binary operations of $O(kn)$ with conventional encoding method [5], [8]. For each constructed test error codeword, calculating Eq. (7) requires real operations of $O(n)$[†]. Therefore, both encoding test error patterns and the real operations dominate mainly the whole decoding complexity. As for space complexity, storing $\tilde{G}$ requires $O(kn)$. Therefore the space complexity is much smaller than that of the other MLD algorithms.

4. Proposed Decoding Algorithm

In the GS decoding algorithm, the complexity of encoding is the same as the conventional encoding method, even if the test error patterns are generated according to the generation rule of test error patterns. In this section, we present an algorithm with low-complexity to construct test error codewords by exploiting both the ordering of test error patterns and the structural property of $\tilde{G}$.

The key ideas of the proposed algorithm are 1) a test error codeword $\tilde{\boldsymbol{w}}_i (= \boldsymbol{t}_i\tilde{G})$ is constructed by adding one or two consecutive rows of $\tilde{G}$ to a test error codeword $\tilde{\boldsymbol{w}}_q$ constructed previously and 2) test error codewords constructed previously are stored in memory in order that we can find $\tilde{\boldsymbol{w}}_q = \boldsymbol{t}_q\tilde{G}$ such that $d_H(\boldsymbol{t}_i, \boldsymbol{t}_q) \leq 2$, where $d_H(\cdot,\cdot)$ denotes the Hamming distance. The first idea is similar to the "metric computation using Gray code ordering" [1]. Gray code ordering has the property in which there is only one difference between the last and the next vector. In the computation method of [1], for obtaining the next metric, only the difference from the previous metric is computed by using the property of Gray code ordering. If the test error pattern is generated in the Gray-code ordering, the next test error codeword is constructed by adding one row of $\tilde{G}$ to the test error codeword lastly constructed. However, we can easily expect that the MRB-based MLD algorithm in which test error patterns are generated in Gray code ordering needs to construct more test error codewords than the GS decoding algorithm. Therefore, we keep the generation rules of test error patterns in the GS decoding algorithm for efficient MLD. In the GS decoding algorithm, the Hamming distance between two test error patterns consecutively generated is more than two in general. Then, based on the second idea mentioned above, we propose the algorithm to construct the next test error codeword effectively in the GS decoding algorithm.

Hereafter we will analyze the GS decoding algorithm in detail. Then we will derive a proposed decoding algorithm which is more efficient than the GS decoding algorithm.

In phase-l reprocessing, let $\boldsymbol{t}_i$ be the current test error pattern and $\boldsymbol{t}_p$ be the last generated test error pattern before generating $\boldsymbol{t}_i$. Let $\boldsymbol{t}_s$ be the next generated test error pattern after generating $\boldsymbol{t}_i$. $\boldsymbol{t}_i$ and $\boldsymbol{t}_s$ may be generated by either the generation rule A or B.

Definition 2: For $\boldsymbol{t}_p$, $\boldsymbol{t}_i \in \mathcal{T}_l$, $i > 2$, we define I_i as the bit position such that $(t_{p,I_i}, t_{p,I_i+1}) = (0, 1)$ and $(t_{i,I_i}, t_{i,I_i+1}) = (1, 0)$. □

Example 3: Figure 1 shows how $\boldsymbol{t}_i$ and I_i change in phase-4 reprocessing. Note that $\boldsymbol{t}_1$ has no I_1 from the definition of I_i. □

i	$\boldsymbol{t}_i$	I_i
1	$000\cdots0001111$	–
2	$000\cdots00\underline{10}111$	$k-4$
3	$000\cdots001\underline{10}11$	$k-3$
4	$000\cdots0011\underline{10}1$	$k-2$
5	$000\cdots00111\underline{10}$	$k-1$
6	$000\cdots0\underline{10}0111$	$k-5$
7	$000\cdots010\underline{10}11$	$k-3$
8	$000\cdots0101\underline{10}1$	$k-2$
9	$000\cdots01011\underline{10}$	$k-1$
10	$000\cdots01\underline{10}011$	$k-4$
⋮	⋮	⋮
13	$000\cdots\underline{10}00111$	$k-6$
14	$000\cdots100\underline{10}11$	$k-3$

Fig. 1 The illustration of the way $\boldsymbol{t}_i$ and I_i change in phase-4 reprocessing.

[†]Real operation means real number addition and its equivalent operations such as additions, subtractions and comparisons.

Lemma 2: For $\boldsymbol{t}_p, \boldsymbol{t}_i \in \mathcal{T}_l$, assume that the next test error pattern $\boldsymbol{t}_s$ is generated. Then $\boldsymbol{t}_s$ has the subsequence such that

$$(t_{s,I_s}, t_{s,I_s+1}, \cdots, t_{s,k}) = (1, 0^{\alpha+1}, 1^{\beta}), \tag{10}$$

where $\alpha \geq 0$ and $\beta \geq 0$. □

Lemma 3: If $\boldsymbol{t}_i, i \geq 2$, is generated, the position I_i is uniquely determined.

Proof: This is implicitly mentioned in p.245, three lines above Eq. (18) of [4]. □

Lemma 4: Assume that $\boldsymbol{t}_p$ is the last generated test error pattern before generating $\boldsymbol{t}_i$, $i \geq 2$. For $\boldsymbol{t}_i$, (t_{i,I_i}, t_{i,I_i+1}) is the rightmost subsequence $(1, 0)$ in $\boldsymbol{t}_i$, i.e.,

$$I_i = \max\{j \mid (t_{i,j}, t_{i,j+1}) = (1, 0)\}, \tag{11}$$

if and only if the position I_i satisfies $(t_{p,I_i}, t_{p,I_i+1}) = (0, 1)$ and $(t_{i,I_i}, t_{i,I_i+1}) = (1, 0)$.

Proof: The if part can be straightforwardly proven by Lemma 2, so we will prove the only if part.

Assuming that there exists a position $I'_i = \max\{j \mid (t_{i,j}, t_{i,j+1}) = (1, 0)\}$ and $(t_{p,I'_i}, t_{p,I'_i+1}) \neq (0, 1)$, we will prove by contradiction. Then there must exist the unique position j', $j' \neq I'_i$, which satisfies $(t_{p,j'}, t_{p,j'+1}) = (0, 1)$ and $(t_{i,j'}, t_{i,j'+1}) = (1, 0)$. By Lemma 2 and 3, we have $(t_{i,j'}, t_{i,j'+1}, \cdots, t_{i,k}) = (1, 0^{\alpha+1}, 1^{\beta})$. This contradicts the assumption that $(t_{i,I'_i}, t_{i,I'_i+1})$ is the rightmost subsequence $(1, 0)$ in $\boldsymbol{t}_i$. Therefore $(t_{p,I'_i}, t_{p,I'_i+1}) = (0, 1)$ and the proof is completed. □

Let $\boldsymbol{t}_p$ be the last generated test error pattern before generating $\boldsymbol{t}_i$, and we now consider encoding $\boldsymbol{t}_i$.

Lemma 5: Assume that (t_{i,I_i}, t_{i,I_i+1}) is the rightmost subsequence $(1, 0)$ in $\boldsymbol{t}_i$. If we need to encode $\boldsymbol{t}_i$ in the GS decoding algorithm, then there exists $\boldsymbol{t}_q \in \mathcal{T}_l$, $q \leq p < i$, given by

$$t_{q,j} = \begin{cases} t_{i,j} \oplus 1, & \text{if } j \in \{I_i, I_i+1\}; \\ t_{i,j}, & \text{otherwise}, \end{cases} \tag{12}$$

and such $\boldsymbol{t}_q$ has been encoded so far in the GS decoding algorithm, i.e.,

$$\Delta(\boldsymbol{t}_i) \leq \underline{L}_{l,i} \implies \Delta(\boldsymbol{t}_q) \leq \underline{L}_{l,q}. \tag{13}$$

Proof: Equation (12) implies $(t_{q,I_i}, t_{q,I_i+1}) = (0, 1)$. Therefore,

$$\begin{aligned} \sum_{j=1}^{k} t_{i,j} 2^{k-j} - \sum_{j=1}^{k} t_{q,j} 2^{k-j} &= t_{i,I_i} 2^{k-I_i} + t_{i,I_i+1} 2^{k-I_i-1} \\ &\quad - t_{q,I_i} 2^{k-I_i} - t_{q,I_i+1} 2^{k-I_i-1} \\ &= 2^{k-I_i} - 2^{k-I_i-1} > 0. \end{aligned} \tag{14}$$

Since test error patterns are generated in the increasing order of standard binary representation, $\boldsymbol{t}_q$ has been already generated when $\boldsymbol{t}_i$ is generated. i.e. $q < i$.

The relationship between $\boldsymbol{t}_i$ and $\boldsymbol{t}_q$ given by Eq. (12) satisfies

$$\begin{aligned} \Delta(\boldsymbol{t}_i) - \Delta(\boldsymbol{t}_q) &= \sum_{j=1}^{k} t_{i,j} |\tilde{\theta}_j| - \sum_{j=1}^{k} t_{q,j} |\tilde{\theta}_j| \\ &\geq |\tilde{\theta}_{I_i}| - |\tilde{\theta}_{I_i+1}| \geq 0. \end{aligned} \tag{15}$$

For $q < i$, since $\underline{L}_{l,q} \geq \underline{L}_{l,i}$, $\boldsymbol{t}_i$ needs to be encoded. At this time, Eq. (13) holds. Since $\boldsymbol{t}_p$ is the last generated test error pattern right before generating $\boldsymbol{t}_i$, $q \leq p < i$ holds. □

Definition 3: Let $\tilde{G}$ be denoted by $\tilde{G} = [\mathcal{I}_k | \tilde{P}]$ where $\mathcal{I}_k$ is the $k \times k$ identity matrix and $\tilde{P}$ is $k \times (n-k)$ matrix. Denote each row of $\tilde{G}$ and $\tilde{P}$ by $\tilde{\boldsymbol{g}}_j$ and $\tilde{\boldsymbol{p}}_j$, $j = 1, 2, \cdots, k$, respectively. i.e.,

$$\tilde{G} = \begin{bmatrix} \tilde{\boldsymbol{g}}_1 \\ \tilde{\boldsymbol{g}}_2 \\ \vdots \\ \tilde{\boldsymbol{g}}_k \end{bmatrix}, \qquad \tilde{P} = \begin{bmatrix} \tilde{\boldsymbol{p}}_1 \\ \tilde{\boldsymbol{p}}_2 \\ \vdots \\ \tilde{\boldsymbol{p}}_k \end{bmatrix}.$$

Furthermore, define $\boldsymbol{f}_j = \tilde{\boldsymbol{g}}_j \oplus \tilde{\boldsymbol{g}}_{j+1}$ and $\boldsymbol{q}_j = \tilde{\boldsymbol{p}}_j \oplus \tilde{\boldsymbol{p}}_{j+1}$ for $j = 1, 2, \cdots, k-1$. □

Assume that we have already obtained $\tilde{\boldsymbol{w}}_q = \boldsymbol{t}_q \tilde{G}$ by encoding $\boldsymbol{t}_q$ where $\boldsymbol{t}_q$ is given by Eq. (12), then we can obtain a test error codeword $\tilde{\boldsymbol{w}}_i$ by

$$\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i \tilde{G} = \boldsymbol{t}_q \tilde{G} \oplus (\boldsymbol{t}_i \oplus \boldsymbol{t}_q) \tilde{G} = \tilde{\boldsymbol{w}}_q \oplus \boldsymbol{f}_{I_i}. \tag{16}$$

Therefore, by Eq. (16), we can construct test error codewords with lower complexity by storing such $\tilde{\boldsymbol{w}}_q$ in memory. The following lemmas help us to find $\tilde{\boldsymbol{w}}_q$ which satisfies Eq. (16) with respect to $\boldsymbol{t}_i$.

Lemma 6: Assume that $\boldsymbol{t}_p$ is the last generated test error pattern right before generating $\boldsymbol{t}_i$. Then equations

$$t_{i,j} = \begin{cases} t_{p,j} \oplus 1, & \text{if } j \in \{I_i, I_i+1\}; \\ t_{p,j}, & \text{otherwise}, \end{cases} \tag{17}$$

and

$$t_{p,j} = t_{i,j} = 1, \quad \text{for } j > I_i + 1, \tag{18}$$

hold for $\boldsymbol{t}_p$ and $\boldsymbol{t}_i$, $i > 2$, if and only if $I_p < I_i$ is satisfied.

Proof: When $I_p < I_i$, we will prove Eqs. (17) and (18). According to the generation rule A or B, $t_{i,j} = t_{p,j}$ holds for $j < I_i$. Therefore, when $I_i = k-1$, Eq. (17) holds. When $I_i \neq k-1$, the proof of Eq. (18) is sufficient. Furthermore by Lemma 4, $(t_{p,I_p}, t_{p,I_p+1}) = (1, 0)$ is the rightmost subsequence $(1, 0)$ in $\boldsymbol{t}_p$.

We here assume that there exists a position $j' >$

$I_i + 1$ satisfying $t_{p,j'} = 0$. Then there exists a position $J' > I_i$ such that $(t_{p,J'}, t_{p,J'+1})$ is the rightmost subsequence $(1,0)$ in $\boldsymbol{t}_p$ since $t_{p,I_i+1} = 1$ by the definition of I_i. This contradicts $I_p < I_i$. Therefore, $t_{p,j} = 1$ for $j > I_i + 1$. According to the generation rule A or B, since $w_H(\boldsymbol{t}_i) = w_H(\boldsymbol{t}_p) = l$ and $\boldsymbol{t}_{i,j} = \boldsymbol{t}_{p,j}$, $\forall j < I_i$,

$$\begin{aligned}\#\{j' | \; t_{i,j'} = 1, j' > I_i + 1\} \\ = \#\{j' | \; t_{p,j'} = 1, j' > I_i + 1\},\end{aligned} \tag{19}$$

must hold for $\boldsymbol{t}_i$, $\boldsymbol{t}_p \in \mathcal{T}_l$ where $\#\{\cdot\}$ represents the cardinality of a set $\{\cdot\}$. Equation (19) implies $t_{i,j'} = t_{p,j'} = 1$, $\forall j' > I_i + 1$. Hence Eqs. (17) and (18) hold and this completes the proof of the if part.

Conversely, we assuming $I_i = k - 1$ and Eq. (17) holds, we will prove $I_p < I_i$. By Lemma 4, (t_{i,I_i}, t_{i,I_i+1}) is the rightmost subsequence $(1,0)$ in $\boldsymbol{t}_i$. Furthermore, Eq. (17) implies $(t_{p,I_i}, t_{p,I_i+1}) = (t_{p,k-1}, t_{p,k}) = (0,1)$. Then I_p is at most $I_i - 1$ and this indicates $I_p < I_i$.

Next, assume $I_i \neq k - 1$ and Eqs. (17) and (18) hold. Then we will prove $I_p < I_i$. By Lemma 4, (t_{i,I_i}, t_{i,I_i+1}) is the rightmost subsequence $(1,0)$ in $\boldsymbol{t}_i$. Furthermore, Eq. (17) implies that the position I_i satisfies $(t_{p,I_i}, t_{p,I_i+1}) = (0,1)$. Therefore by Eq. (18), the rightmost subsequence $(t_{p,I_p}, t_{p,I_p+1}) = (1,0)$ must satisfy $I_p < I_i$. Hence the proof is completed. □

Lemma 7: Assume that $\boldsymbol{t}_p$ is the last generated test error pattern right before generating $\boldsymbol{t}_i$. If $I_i < I_p$, then $\boldsymbol{t}_i$, $i > 2$, satisfies $t_{i,I_i+2} = 0$.

Proof: By the definition of I_i, $(t_{p,I_i}, t_{p,I_i+1}) = (0,1)$. By Lemma 4, $(t_{p,I_p}, t_{p,I_p+1}) = (1,0)$ is the rightmost subsequence $(1,0)$ in $\boldsymbol{t}_p$. Therefore, $\boldsymbol{t}_p$ has at least one position satisfying $t_{p,j} = 0$, $\exists j > I_i + 1$, in the case $I_i < I_p$. According to the generation rule A or B, since $t_{i,j} = t_{p,j}$, $\forall j < I_i$, Eq. (19) must hold for $\boldsymbol{t}_i, \boldsymbol{t}_p \in \mathcal{T}_l$. Since $(t_{i,I_i+2}, t_{i,I_i+3}, \cdots, t_{i,k}) = (0^\alpha, 1^\beta)$ by Lemma 2, $\alpha \geq 1$ holds by the fact that there is at least one position such that $t_{p,j} = 0$, $\exists j > I_i + 1$. Therefore $t_{i,I_i+2} = 0$ holds for $\boldsymbol{t}_i$. □

Lemma 8: If $I_i < I_p$, $i > 2$, then $I_q = I_i + 1$ for $\boldsymbol{t}_q$ given by Eq. (12).

Proof: Since $(t_{i,I_i+2}, t_{i,I_i+3}, \cdots, t_{i,k}) = (0^\alpha, 1^\beta)$, $\alpha \geq 1$, by Lemma 2 and 7, $\boldsymbol{t}_q$ has the subsequence $(t_{q,I_i+1}, t_{q,I_i+2}, \cdots, t_{q,k}) = (1, 0^\alpha, 1^\beta)$, $\alpha \geq 1$. Therefore $(t_{q,I_i+1}, t_{q,I_i+2}) = (1,0)$ is the rightmost subsequence (1,0) in $\boldsymbol{t}_q$. This indicates $I_q = I_i + 1$ by Lemma 4 and the proof is completed. □

Remark 1: In phase-l reprocessing, if $p = 1$ and $i = 2$, then $I_p = I_1$ does not exist. However, in this case, Eq. (17) holds. This fact must be reminded to derive the low-complexity method for constructing test error codeword in the sequel. □

Theorem 1: Assume that $\boldsymbol{t}_p$ is the last generated test error pattern before generating $\boldsymbol{t}_i$. If $I_p < I_i$, $i > 2$, then

$$\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i \tilde{G} = \tilde{\boldsymbol{w}}_p \oplus \boldsymbol{f}_{I_i}, \tag{20}$$

where $\boldsymbol{f}_{I_i} = \tilde{\boldsymbol{g}}_{I_i} \oplus \tilde{\boldsymbol{g}}_{I_i+1}$.

If $i = 2$ or $I_i < I_p$, $i > 2$, then

$$\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i \tilde{G} = \tilde{\boldsymbol{w}}_q \oplus \boldsymbol{f}_{I_i}, \tag{21}$$

where $\tilde{\boldsymbol{w}}_q = \boldsymbol{t}_q \tilde{G}$ and $\boldsymbol{t}_q$ is given by Eq. (12).

Proof: (case $I_p < I_i$) Let $\boldsymbol{b} = (b_1, b_2, \cdots, b_k) \in \{0,1\}^k$ be

$$b_j = \begin{cases} 1, & \text{if } j \in \{I_i, I_i + 1\}; \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

By Eq. (17) of Lemma 6,

$$\begin{aligned}\tilde{\boldsymbol{w}}_i &= \boldsymbol{t}_i \tilde{G} = (\boldsymbol{t}_p \oplus \boldsymbol{b}) \tilde{G} \\ &= \boldsymbol{t}_p \tilde{G} \oplus \boldsymbol{b} \tilde{G} = \tilde{\boldsymbol{w}}_p \oplus \boldsymbol{f}_{I_i}.\end{aligned} \tag{23}$$

(case $I_i < I_p$) For $\boldsymbol{t}_i$, $i > 2$, there exists $\boldsymbol{t}_q$ satisfying Eq. (12). By Eq. (12), we have

$$\begin{aligned}\tilde{\boldsymbol{w}}_i &= \boldsymbol{t}_i \tilde{G} = (\boldsymbol{t}_q \oplus \boldsymbol{b}) \tilde{G} \\ &= \boldsymbol{t}_q \tilde{G} \oplus \boldsymbol{b} \tilde{G} = \tilde{\boldsymbol{w}}_q \oplus \boldsymbol{f}_{I_i},\end{aligned} \tag{24}$$

where $\boldsymbol{b}$ is given by Eq. (22).

For $i = 2$, $\boldsymbol{t}_p = \boldsymbol{t}_1$ and $\boldsymbol{t}_i = \boldsymbol{t}_2$ have the relationship given by Eq. (17), (see remark 1). Therefore, we have $\boldsymbol{t}_p = \boldsymbol{t}_q$ and Eq. (21) holds.

The proof is completed. □

By the following lemma, we can effectively calculate $\Delta(\boldsymbol{t}_i)$. We define $\delta_j = |\tilde{\theta}_j| - |\tilde{\theta}_{j+1}|$, $j = 1, 2, \cdots, k-1$.

Lemma 9: If $I_p < I_i$, $i > 2$, then we have

$$\Delta(\boldsymbol{t}_i) = \Delta(\boldsymbol{t}_p) + \delta_{I_i}. \tag{25}$$

If $i = 2$ or $I_i < I_p$, $i > 2$, then we have

$$\Delta(\boldsymbol{t}_i) = \Delta(\boldsymbol{t}_q) + \delta_{I_i}. \tag{26}$$

□

By using $\Delta(\boldsymbol{t}_i)$, we have the following lemma.

Lemma 10: The function $\Lambda(\tilde{\boldsymbol{w}}_i)$ can be rewritten as

$$\Lambda(\tilde{\boldsymbol{w}}_i) = \Delta(\boldsymbol{t}_i) + \sum_{j=k+1}^{n} (-1)^{\tilde{z}_j \oplus \tilde{c}_{0,j}} \tilde{w}_{i,j} |\tilde{\theta}_j|, \tag{27}$$

where $\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i \tilde{G}$.

Proof: In the right hand side of Eq. (7), $\tilde{z}_j \oplus \tilde{c}_{0,j} = 0$ holds for $1 \leq j \leq k$ since $\tilde{\boldsymbol{c}}_0 = \boldsymbol{u}\tilde{G} = (\tilde{z}_1, \tilde{z}_2, \cdots, \tilde{z}_k)\tilde{G}$. Furthermore, $\tilde{w}_{i,j} = t_{i,j}$ for $1 \leq j \leq k$ since $\tilde{G} =$

$[\mathcal{I}_k|\tilde{P}]$ is the systematic generator matrix. Then Eq. (7) expands as

$$\begin{aligned}\Lambda(\tilde{\boldsymbol{w}}_i) &= \sum_{j=1}^{n}(-1)^{\tilde{z}_j\oplus\tilde{c}_{0,j}}\tilde{w}_{i,j}|\tilde{\theta}_j| \\ &= \sum_{j=1}^{k}\tilde{w}_{i,j}|\tilde{\theta}_j| + \sum_{j=k+1}^{n}(-1)^{\tilde{z}_j\oplus\tilde{c}_{0,j}}\tilde{w}_{i,j}|\tilde{\theta}_j| \\ &= \Delta(\boldsymbol{t}_i) + \sum_{j=k+1}^{n}(-1)^{\tilde{z}_j\oplus\tilde{c}_{0,j}}\tilde{w}_{i,j}|\tilde{\theta}_j|. \qquad (28)\end{aligned}$$

Hence we have Eq. (27) and the proof is completed. □

When we calculate $\Lambda(\cdot)$ by Eq. (27), the number of real operations is no more than that by calculation of Eq. (7).

Definition 4: Define $\mathcal{T}_l^{(j)}$ as the set of test error patterns $\boldsymbol{t}_\tau \in \mathcal{T}_l$ which satisfies $I_\tau = j$, i.e., $\mathcal{T}_l^{(j)} = \{\boldsymbol{t}_\tau \in \mathcal{T}_l \mid I_\tau = j\}$. □

Lemma 11: Assume that $I_i < I_p$ and $\boldsymbol{t}_q$ is given by Eq. (12). Then $\boldsymbol{t}_q$ is the most recently generated element in $\mathcal{T}_l^{(I_i+1)}$ before generating $\boldsymbol{t}_i$. i.e.,

$$q = \max\{\tau \mid \boldsymbol{t}_\tau \in \mathcal{T}_l^{(I_i+1)}, \tau < i\}. \qquad (29)$$

Proof: Assume that there exists $\boldsymbol{t}_{q'} \in \mathcal{T}_l^{(I_i+1)}$, $q < q' < i$. If there exists a position $j < I_i$ such that $t_{q',j} \neq t_{q,j}$, then either $q' < q$ or $i < q'$ holds. Therefore $t_{q',j} = t_{q,j} = t_{i,j}$ for all $j < I_i$. By Lemma 2, $(t_{q,I_i+1}, t_{q,I_i+2}, \cdots, t_{q,k}) = (1, 0^{\alpha+1}, 1^\beta)$ and $(t_{q',I_i+1}, t_{q',I_i+2}, \cdots, t_{q',k}) = (1, 0^{\alpha'+1}, 1^{\beta'})$. If $t_{q',I_i} = 1$, then $q' > i$ hence $t_{q',I_i} = t_{q,I_i} = 0$. Since $w_H(\boldsymbol{t}_q) = w_H(\boldsymbol{t}_{q'})$, we have $\#\{j \mid t_{q,j} = 1, j > I_i\} = \#\{j \mid t_{q',j} = 1, j > I_i\}$. Therefore $\beta = \beta'$ and this indicates $q' = q$. This is contradiction and Eq. (29) holds. □

We now describe the method for finding $\tilde{\boldsymbol{w}}_p = \boldsymbol{t}_p\tilde{G}$ in Eq. (20) or $\tilde{\boldsymbol{w}}_q = \boldsymbol{t}_q\tilde{G}$ in Eq. (21) fast. First, k arrays $\tilde{\boldsymbol{w}}^{(j)}$ of length n, $1 \le j \le k$, are prepared. In phase-l reprocessing of the GS decoding algorithm, we store test error codeword $\tilde{\boldsymbol{w}}_\tau = \boldsymbol{t}_\tau\tilde{G}$, which has been already constructed before constructing $\tilde{\boldsymbol{w}}_i$, into the array $\tilde{\boldsymbol{w}}^{(I_\tau)}$, i.e., $\tilde{\boldsymbol{w}}^{(I_\tau)} := \tilde{\boldsymbol{w}}_\tau$. Similarly, $\Delta^{(j)}$, $1 \le j \le k$, are prepared. In phase-l reprocessing, we store $\Delta(\boldsymbol{t}_\tau)$ into the array $\Delta^{(I_\tau)}$, i.e., $\Delta^{(I_\tau)} := \Delta(\boldsymbol{t}_\tau)$. Note that a superscript of arrays $\tilde{\boldsymbol{w}}^{(I_\tau)}$ and $\Delta^{(I_\tau)}$ which means the address of memory is uniquely determined by $\boldsymbol{t}_\tau$ (or equivalently by $\boldsymbol{w}_\tau$). $\tilde{\boldsymbol{w}}^{(j)}$ plays a role as $\tilde{\boldsymbol{w}}_p$ of Eq. (20) or $\tilde{\boldsymbol{w}}_q$ of Eq. (21) in the proposed decoding algorithm presented below. In the same way, $\Delta^{(j)}$ plays a role as $\Delta(\boldsymbol{t}_p)$ or $\Delta(\boldsymbol{t}_q)$. Furthermore, we now rewrite $\boldsymbol{t}_1 \in \mathcal{T}_{l-1}$ as $\boldsymbol{t}_1^{(l-1)}$ and rewrite $\boldsymbol{t}_1 \in \mathcal{T}_l$ as $\boldsymbol{t}_1^{(l)}$. Let $\tilde{\boldsymbol{w}}'$ be the stored test error codeword such that $\tilde{\boldsymbol{w}}' = \boldsymbol{t}_1^{(l-1)}\tilde{G}$. Let $\Delta' = \Delta(\boldsymbol{t}_1^{(l-1)})$ be also stored. Remark that $\boldsymbol{t}_1^{(l-1)}$ and $\boldsymbol{t}_1^{(l)}$ have only one different symbol in the position $j = k - l + 1$. When phase-$(l-1)$ reprocessing is terminated and phase-l reprocessing is started, test error codeword $\tilde{\boldsymbol{w}}_1 = \boldsymbol{t}_1^{(l)}\tilde{G}$ can be obtained by $\tilde{\boldsymbol{w}}_1 = \tilde{\boldsymbol{w}}' \oplus \boldsymbol{g}_{k-l+1}$. Similarly, we can calculate $\Delta(\boldsymbol{t}_1^{(l)})$ in the way $\Delta(\boldsymbol{t}_1^{(l)}) = \Delta' + |\tilde{\theta}_{k-l+1}|$.

We here propose an improved version of the GS decoding algorithm below, using low-complexity encoding method, where we define $I_1 = k - l - 1$.

[Proposed decoding algorithm]

1) Generate $\tilde{\boldsymbol{c}}_0 := \boldsymbol{u}\tilde{G}$, and set $\underline{L} := L(\tilde{\boldsymbol{c}}_0)$, $\tilde{\boldsymbol{w}}' := \boldsymbol{0}$, $\tilde{\boldsymbol{w}}^* := \boldsymbol{0}$, $\Delta' := 0$, $\underline{\Lambda} := 0$ and $l := 1$.
2) a) Generate $\boldsymbol{t}_1 \in \mathcal{T}_l$, and set $I_1 := k - l + 1$, $\Delta' := \Delta' + |\tilde{\theta}_{I_1}|$ and $\Delta^{(I_1)} := \Delta'$. If $\underline{L} \le \Delta'$, then output $\tilde{\boldsymbol{c}}_{ML} := \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}^*$ and stop, otherwise $\tilde{\boldsymbol{w}}' := \tilde{\boldsymbol{w}}' \oplus \tilde{\boldsymbol{g}}_{I_1}$, and set $\tilde{\boldsymbol{w}}^{(I_1)} := \tilde{\boldsymbol{w}}'$.
   b) Calculate $\Lambda(\tilde{\boldsymbol{w}}^{(I_1)})$. If $\Lambda(\tilde{\boldsymbol{w}}^{(I_1)}) \le \underline{\Lambda}$, then $\underline{\Lambda} := \Lambda(\tilde{\boldsymbol{w}}^{(I_1)})$, $\underline{L} := L(\tilde{\boldsymbol{c}}_0) + \underline{\Lambda}$ and $\tilde{\boldsymbol{w}}^* := \tilde{\boldsymbol{w}}^{(I_1)}$.
3) Set $p := 1$, $i := 2$, and generate $\boldsymbol{t}_i$.
   a) Find the position I_i from $\boldsymbol{t}_i$. If $I_p < I_i$, then
   $$\Delta^{(I_i)} := \Delta^{(I_p)} + \delta_{I_i}, \qquad (30)$$
   otherwise
   $$\Delta^{(I_i)} := \Delta^{(I_i+1)} + \delta_{I_i}. \qquad (31)$$
   b) If $\underline{L} \le \Delta^{(I_i)}$, then set $I_p := I_i$ and try to generate $\boldsymbol{t}_s$ by the generation rule B, otherwise go to 3-c). If there exists $\boldsymbol{t}_s$, then set $\boldsymbol{t}_p := \boldsymbol{t}_i$, $\boldsymbol{t}_i := \boldsymbol{t}_s$ and go to 3-a), otherwise go to 4).
   c) If $I_p < I_i$, then
   $$\tilde{\boldsymbol{w}}^{(I_i)} := \tilde{\boldsymbol{w}}^{(I_p)} \oplus \boldsymbol{f}_{I_i}, \qquad (32)$$
   otherwise
   $$\tilde{\boldsymbol{w}}^{(I_i)} := \tilde{\boldsymbol{w}}^{(I_i+1)} \oplus \boldsymbol{f}_{I_i}. \qquad (33)$$
   d) Set $I_p := I_i$ and calculate $\Lambda(\tilde{\boldsymbol{w}}^{(I_i)})$. If $\Lambda(\tilde{\boldsymbol{w}}^{(I_i)}) \le \underline{\Lambda}$, then set $\underline{\Lambda} := \Lambda(\tilde{\boldsymbol{w}}^{(I_i)})$, $\underline{L} := L(\tilde{\boldsymbol{c}}_0) + \underline{\Lambda}$ and $\tilde{\boldsymbol{w}}^* := \tilde{\boldsymbol{w}}^{(I_i)}$. Set $\boldsymbol{t}_p := \boldsymbol{t}_i$, $i := i + 1$, generate $\boldsymbol{t}_i$ by the generation rule A and go to 3-a).
4) Set $l := l + 1$. If $l \le \kappa$, then go to 2), otherwise output $\tilde{\boldsymbol{c}}_{ML} := \tilde{\boldsymbol{c}}_0 \oplus \tilde{\boldsymbol{w}}^*$ and stop. □

In order to show the validity of the proposed decoding algorithm, we derive the following theorem.

Theorem 2: Let

$$J = \begin{cases} I_p, & \text{if } I_p < I_i; \\ I_i + 1, & \text{otherwise.} \end{cases} \qquad (34)$$

Then in step 3-c) of the proposed decoding algorithm,

$$\tilde{\boldsymbol{w}}^{(I_i)} = \tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i\tilde{G} = \tilde{\boldsymbol{w}}^{(J)} \oplus \boldsymbol{f}_{I_i}, \quad (35)$$

where $\boldsymbol{f}_{I_i} = \tilde{\boldsymbol{g}}_{I_i} \oplus \tilde{\boldsymbol{g}}_{I_i+1}$. Furthermore, in step 3-a) of the proposed decoding algorithm,

$$\Delta^{(I_i)} = \Delta(\boldsymbol{t}_i) = \Delta^{(J)} + \delta_{I_i}, \quad (36)$$

where $\delta_{I_i} = |\tilde{\theta}_{I_i}| - |\tilde{\theta}_{I_i+1}|$.

Proof: We will prove Eq. (35) and Eq. (36) by mathematical induction.

For $i = 2$, $\boldsymbol{t}_p = \boldsymbol{t}_1$ and $\boldsymbol{t}_i = \boldsymbol{t}_2$ have the relationship given by Eq. (17), (see remark 1). Furthermore, since I_1 is defined as $I_1 = k-l+1$, $I_1 = I_2+1$ holds. Therefore, Eq. (35) and Eq. (36) hold.

We here assume $\tilde{\boldsymbol{w}}^{(I_p)} = \tilde{\boldsymbol{w}}_p = \boldsymbol{t}_p\tilde{G}$ and $\Delta^{(I_p)} = \Delta(\boldsymbol{t}_p)$. Consider that $\boldsymbol{t}_i$ satisfying $I_p < I_i$ is the next generated test error pattern after $\boldsymbol{t}_p$. By Lemma 6, $\boldsymbol{t}_p$ and $\boldsymbol{t}_i$ has the relationship given by Eq. (17). Then by Eq. (20) of Theorem 1,

$$\tilde{\boldsymbol{w}}^{(I_i)} = \boldsymbol{t}_i\tilde{G} = \boldsymbol{t}_p\tilde{G} \oplus \boldsymbol{f}_{I_i} = \tilde{\boldsymbol{w}}^{(I_p)} \oplus \boldsymbol{f}_{I_i}. \quad (37)$$

Furthermore, by Lemma 9, we have

$$\Delta^{(I_i)} = \Delta(\boldsymbol{t}_i) = \Delta(\boldsymbol{t}_p) + \delta_{I_i} = \Delta^{(I_p)} + \delta_{I_i}. \quad (38)$$

Next, we consider the case that $I_i < I_p$. For $\boldsymbol{t}_i$, $i > 2$, there exists $\boldsymbol{t}_q$ satisfying Eq. (12). Assume $\tilde{\boldsymbol{w}}^{(I_q)} = \tilde{\boldsymbol{w}}_q = \boldsymbol{t}_q\tilde{G}$ and $\Delta^{(I_q)} = \Delta(\boldsymbol{t}_q)$. By Lemma 8, we have $I_q = I_i + 1$. Therefore, by Eq. (21) of Theorem 1,

$$\tilde{\boldsymbol{w}}^{(I_i)} = \boldsymbol{t}_i\tilde{G} = \boldsymbol{t}_q\tilde{G} \oplus \boldsymbol{f}_{I_i} = \tilde{\boldsymbol{w}}^{(I_i+1)} \oplus \boldsymbol{f}_{I_i}. \quad (39)$$

Furthermore, $\boldsymbol{t}_q$ given by Eq. (12) is the most recently generated element in $\mathcal{T}_l^{(I_i+1)}$ by Lemma 11, then no other $\tilde{\boldsymbol{w}}_{q'} = \boldsymbol{t}_{q'}\tilde{G}$ such that $\boldsymbol{t}_{q'} \in \mathcal{T}_l^{(I_i+1)}$, $q' \neq q$, have been stored in $\tilde{\boldsymbol{w}}^{(I_q)}$ when $\boldsymbol{t}_i$ is generated. Similarly, $\Delta(\boldsymbol{t}_q)$ has been stored into $\Delta^{(I_q)}$ when $\boldsymbol{t}_i$ is generated. Consequently, the validity of Eq. (35) and Eq. (36) are guaranteed. The proof is completed. □

Theorem 3: The time complexity of constructing a test error codeword in the proposed decoding algorithm is $O(n)$.

Proof: Equations (32) and (33) indicate the time complexity to construct a test error codeword requires n binary exclusive OR operations. Therefore the time complexity of constructing a test error codeword is $O(n)$. □

Note that the time complexity of constructing a test error codeword in the GS decoding algorithm is $O(kn)$.

The main purpose of the proposed decoding algorithm is to reduce the time complexity of constructing test error codewords in the GS decoding algorithm. In addition to the above improvement of Theorem 3, we can reduce the number of real operations in the GS decoding algorithm.

Theorem 4: In the proposed decoding algorithm, the number of real operations is no more than that in the GS decoding algorithm. In phase-l reprocessing for $l \geq 2$, the proposed decoding algorithm always reduces the number of real operations of the GS decoding algorithm.

Proof: In phase-l reprocessing, the calculation of the equation $\Delta(\boldsymbol{t}_i) = \sum_{j=1}^{k} t_{i,j}|\tilde{\theta}_j|$, which is employed in the GS decoding algorithm, requires $l-1$ real operations. On the other hand, the calculation of Eq. (25) or (26), which is employed in the proposed decoding algorithm, requires only 1 real operation.

Furthermore, the calculation of Eq. (7) with respect to $\tilde{\boldsymbol{w}}_i$, which is employed in the GS decoding algorithm, requires $w_H(\tilde{\boldsymbol{w}}_i) - 1$ real operations. On the other hand, the calculation of Eq. (27) with respect to $\tilde{\boldsymbol{w}}_i$, which is employed in the proposed decoding algorithm, requires $w_H(\tilde{\boldsymbol{w}}_i) - l$ real operations.

Therefore the number of the real operations in the proposed decoding algorithm is at most the same as that in the GS decoding algorithm. In phase-l reprocessing for $l \geq 2$, the use of Lemma 9 and Eq. (27) guarantees the reduction of the number of the real operations in the proposed decoding algorithm. □

Theorem 5: The additional space complexity to the GS decoding algorithm is $O(kn)$. □

Proof: The proposed decoding algorithm requires the space complexity for k $\tilde{\boldsymbol{w}}^{(I_i)}$s and $\tilde{\boldsymbol{w}}'$. Then we store at most $k+1$ test error codewords in memory. This space complexity is $O(kn)$. In order to construct test error codeword fast, the proposed decoding algorithm utilizes $\boldsymbol{f}_j = \boldsymbol{g}_j \oplus \boldsymbol{g}_{j+1}$, $j = 1, 2, \cdots, k-1$. The space complexity to store $\boldsymbol{f}_j$, $j = 1, 2, \cdots, k-1$, is $O(kn)$. In addition to the above space complexity, the rest of space complexity increased by the proposed decoding algorithm is for k $\Delta^{(I_i)}$s and Δ'. Therefore, the proposed decoding algorithm needs space complexity of $O(kn)$ besides the space complexity which the GS decoding algorithm needs. □

Remark 2: The space complexity for storing the generator matrix is $O(k(n-k))$, which is the space complexity of the GS decoding algorithm. Therefore, Theorem 5 implies that the space complexity of the proposed decoding is the same order as that of the GS decoding algorithm. □

We now consider further reduction in the time and space complexity to construct $\tilde{\boldsymbol{w}}_i$ by Eq. (20) or Eq. (21) of Theorem 1, by the structural property of $\tilde{G}$. In phase-l reprocessing, for $i = 1, \cdots, \binom{k}{l}$, we define $(n-k)$-tuple $\tilde{\boldsymbol{v}}_i$ such that $\tilde{\boldsymbol{v}}_i = \boldsymbol{t}_i \tilde{P} = (\tilde{w}_{i,k+1}, \tilde{w}_{i,k+2}, \cdots, \tilde{w}_{i,n})$ where $\tilde{P}$ is defined by definition 3. Then $\tilde{\boldsymbol{w}}_i$ is expressed by $\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i \circ \tilde{\boldsymbol{v}}_i$, where $\boldsymbol{t} \circ \tilde{\boldsymbol{v}}$ denotes the concatenation of $\boldsymbol{t}$ and $\tilde{\boldsymbol{v}}$. By Theorem 1, we derive the following theorem without proof.

Theorem 6: If $I_p < I_i$, $i > 2$, then

$$\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i \circ (\tilde{\boldsymbol{v}}_p \oplus \boldsymbol{q}_{I_i}), \tag{40}$$

where $\boldsymbol{q}_{I_i} = \tilde{\boldsymbol{p}}_{I_i} \oplus \tilde{\boldsymbol{p}}_{I_i+1}$.

If $I_i < I_p$, $i \geq 2$, then

$$\tilde{\boldsymbol{w}}_i = \boldsymbol{t}_i \circ (\tilde{\boldsymbol{v}}_q \oplus \boldsymbol{q}_{I_i}), \tag{41}$$

where $\tilde{\boldsymbol{v}}_q = \boldsymbol{t}_q \tilde{P}$ and $\boldsymbol{t}_q$ is given by Eq. (12). □

Theorem 6 implies that at most $(n-k)$ binary operations is required to construct a test error codeword by storing $(n-k)$-tuples $\tilde{\boldsymbol{v}}_q$ in memory. This complexity is smaller than that of Eq. (20) or Eq. (21) of Theorem 1, which requires at most n binary operations, and much smaller than that of the conventional encoding method.

Remark 3: Remark that it is sufficient to store $(n-k)$-tuples $\tilde{\boldsymbol{v}}_q$ instead of storing n-tuples $\tilde{\boldsymbol{w}}_q$. This space complexity for storing $\tilde{\boldsymbol{v}}_q$ is only $O(k(n-k))$. Furthermore, the space complexity for storing $\boldsymbol{q}_j$, $j = 1, 2, \cdots, k-1$, is also $O(k(n-k))$. Therefore, the additional space complexity to the GS decoding algorithm is $O(k(n-k))$. □

Remark 4: In phase-l reprocessing of the GS decoding algorithm, the number of binary operations required for constructing a test error codeword is ln when l rows of $\tilde{G}$ are simply added. However, since $\tilde{G}$ is systematic, the number of binary operations for a test error codeword is only $l(n-k)$, if we consider that l rows of Q are added. In this case, the number of binary operations for a test error codeword also depends on l. On the contrary, in the proposed decoding algorithm, the number of binary operations required for constructing a test error codeword is $n-k$, which is independent of l.

In the GS decoding algorithm, the number of real operations for calculating $\Lambda(\tilde{\boldsymbol{w}}_i)$ is $w_H(\tilde{\boldsymbol{w}}_i) - 1$. On the contrary, in the proposed decoding algorithm, the number of real operations for calculating $\Lambda(\tilde{\boldsymbol{w}}_i)$ is $w_H(\tilde{\boldsymbol{w}}_i) - l$.

Therefore, the time complexity in the proposed decoding algorithm is equal to or smaller than that of the GS decoding algorithm for one test error codeword. Furthermore, in phase-l reprocessing for $l \geq 2$, the time complexity in the proposed decoding algorithm is always smaller than that of the GS decoding algorithm. □

Theorem 7: The proposed decoding algorithm performs MLD.

Proof: Theorem 2 guarantees that test error codewords constructed in the proposed decoding algorithm are the same as that in the GS decoding algorithm. Since the GS decoding algorithm performs MLD, the proposed decoding algorithm always finds the ML codeword. □

5. Simulation Results

In this section, we present simulation results for the binary (63,30,13) BCH code and the binary (127,64,21) BCH code. We compare the proposed decoding algorithm with the GS decoding algorithm. The results are obtained by simulating 10000 codewords for each signal to noise ratio (E_b/N_0 [dB]) and the average values are shown in tables. In the tables, we use the following notation.

B_{GS}: the number of binary operations of constructing test error codewords in the GS decoding algorithm

B_{pro}: the number of binary operations of constructing test error codewords in the proposed decoding algorithm

R_B: the ratio of binary operations in the proposed decoding algorithm to the GS decoding algorithm, i.e., $R_B = B_{pro}/B_{GS}$

RO_{GS}: the number of real operations in the GS decoding algorithm

RO_{pro}: the number of real operations in the proposed decoding algorithm

R_{RO}: the ratio of real operations in the proposed decoding algorithm to the GS decoding algorithm, i.e., $R_{RO} = RO_{pro}/RO_{GS}$

In Tables 1 and 2, the numbers of binary operations of constructing test error codewords for each decoding algorithm are shown. In phase-l reprocessing, the number of binary operations for constructing a test error code-

Table 1 The number of binary operations to construct test error codewords for the $(63, 30, 13)$ BCH code.

E_b/N_0	B_{GS}	B_{pro}	R_B
1.50	$1.76 \cdot 10^8$	$5.11 \cdot 10^8$	0.2896
2.00	$8.76 \cdot 10^8$	$2.67 \cdot 10^8$	0.3044
2.50	$3.80 \cdot 10^8$	$1.24 \cdot 10^8$	0.3273
3.00	$1.45 \cdot 10^8$	$5.28 \cdot 10^7$	0.3647
3.50	$4.69 \cdot 10^7$	$2.01 \cdot 10^7$	0.4293
4.00	$1.57 \cdot 10^7$	$7.90 \cdot 10^6$	0.5024
4.50	$5.85 \cdot 10^6$	$3.27 \cdot 10^6$	0.5594
5.00	$1.73 \cdot 10^6$	$1.23 \cdot 10^6$	0.7065
5.50	$5.62 \cdot 10^5$	$4.68 \cdot 10^5$	0.8329

Table 2 The number of binary operations to construct test error codewords for the (127, 64, 21) BCH code.

E_b/N_0	B_{GS}	B_{pro}	R_B
2.50	$4.50 \cdot 10^{12}$	$7.05 \cdot 10^{11}$	0.1566
3.00	$6.40 \cdot 10^{11}$	$1.12 \cdot 10^{11}$	0.1757
3.50	$5.95 \cdot 10^{10}$	$1.29 \cdot 10^{10}$	0.2176
4.00	$6.11 \cdot 10^{9}$	$1.69 \cdot 10^{9}$	0.2773
4.50	$7.18 \cdot 10^{8}$	$2.61 \cdot 10^{8}$	0.3644
5.00	$1.03 \cdot 10^{8}$	$4.99 \cdot 10^{7}$	0.4845
5.50	$1.82 \cdot 10^{7}$	$1.17 \cdot 10^{7}$	0.6445
6.00	$3.86 \cdot 10^{6}$	$3.12 \cdot 10^{6}$	0.8077
6.50	$8.39 \cdot 10^{5}$	$8.21 \cdot 10^{5}$	0.9789

Table 3 The number of real operations for the (63, 30, 13) BCH code.

E_b/N_0	RO_{GS}	RO_{pro}	R_{RO}
1.50	$4.35 \cdot 10^{4}$	$3.41 \cdot 10^{4}$	0.7843
2.00	$2.26 \cdot 10^{4}$	$1.80 \cdot 10^{4}$	0.7965
2.50	$1.07 \cdot 10^{4}$	$8.68 \cdot 10^{3}$	0.8113
3.00	$4.72 \cdot 10^{3}$	$3.98 \cdot 10^{3}$	0.8433
3.50	$2.06 \cdot 10^{3}$	$1.83 \cdot 10^{3}$	0.8884
4.00	$1.10 \cdot 10^{3}$	$1.02 \cdot 10^{3}$	0.9334
4.50	$7.32 \cdot 10^{2}$	$7.06 \cdot 10^{2}$	0.9645
5.00	$5.61 \cdot 10^{2}$	$5.54 \cdot 10^{2}$	0.9877
5.50	$4.78 \cdot 10^{2}$	$4.76 \cdot 10^{2}$	0.9958

Table 4 The number of real operations for the (127, 64, 21) BCH code.

E_b/N_0	RO_{GS}	RO_{pro}	R_{RO}
2.50	$5.43 \cdot 10^{7}$	$4.09 \cdot 10^{7}$	0.7542
3.00	$8.41 \cdot 10^{6}$	$6.53 \cdot 10^{6}$	0.7765
3.50	$9.24 \cdot 10^{5}$	$7.53 \cdot 10^{5}$	0.8150
4.00	$1.16 \cdot 10^{5}$	$9.96 \cdot 10^{4}$	0.8571
4.50	$1.83 \cdot 10^{4}$	$1.64 \cdot 10^{4}$	0.8960
5.00	$4.32 \cdot 10^{3}$	$4.07 \cdot 10^{3}$	0.9421
5.50	$1.87 \cdot 10^{3}$	$1.83 \cdot 10^{3}$	0.9791
6.00	$1.29 \cdot 10^{3}$	$1.29 \cdot 10^{3}$	0.9946
6.50	$1.09 \cdot 10^{3}$	$1.08 \cdot 10^{3}$	0.9987

word is calculated as $l(n-k)$ and $(n-k)$ in the GS decoding and the proposed decoding algorithm, respectively. In Tables 3 and 4, the number of real operations for each decoding algorithm are shown. For sorting the columns of generator matrix, the quick sort technique, whose time complexity is $O(n \log n)$, is used. In this paper, the number of real operations for calculating $L(\tilde{\boldsymbol{c}}_0)$ is counted as $d_H(\tilde{\boldsymbol{z}}, \tilde{\boldsymbol{c}}_0) - 1$ instead of $n-1$.

By Table 1 for the (63, 30, 13) BCH code, the number of binary operations in the proposed decoding algorithm is 4/5 ones in the GS decoding algorithm at 5.5 [dB] where R_B is maximum. From 3.0 to 1.5 [dB], the proposed decoding algorithm reduces them up to 1/3. By Table 2 for the (127, 64, 21) BCH code, the number of binary operations in the proposed decoding algorithm is almost the same as that in the GS decoding algorithm at high SNR. At 6.5 [dB], about 98% of test error patterns encoded in total are in phase-1 reprocessing and the rest 2% of them are in phase-2 reprocessing. However, as SNR becomes lower, R_B decreases and the proposed decoding algorithm reduces the number of binary operations up to 1/5 at 2.5 [dB]. Tables 1 and 2 show that R_B monotonically decreases as SNR becomes smaller. The reason is that the weight of test error patterns, l, generated in the algorithm tends to be larger at the low SNR. In this case, since the number of binary operations for a test error codeword depends on l in the GS decoding algorithm and is independent of l in the proposed decoding algorithm, difference between B_{GS} and B_{pro} becomes larger. Note also that the value R_B for the (127, 64, 21) code is smaller than that for the (63, 30, 13) code at each SNR.

Although details are omitted here, for other codes with length $n = 63$, the simulation results of R_B for the (63, 24, 15) BCH and the (63, 36, 11) BCH codes at 1.5 [dB] are 0.2532 and 0.3426, respectively. For other codes with length $n = 127$, the simulation results of R_B for the (127, 50, 27) BCH and the (127, 78, 15) BCH codes at 3.0 [dB] are 0.1369 and 0.1791, respectively.

By Tables 3 and 4, the value R_{RO} for the (127, 64, 21) code is always smaller than that for the (63, 30, 13) code at each SNR. However, the number of real operations for sorting columns of generator matrix is about 490 for the (63, 30, 13) code and about 1140 for the (127, 64, 21) code on average although these numbers slightly change as SNR changes. Note that the number of real operations for sorting costs the same for both the GS and the proposed decoding algorithm. Then at 5.5–4.5 [dB], the number of real operations for sorting columns of generator matrix is dominant in the total number of real operations for the (63, 30, 13) code. Similarly, at 6.5–5.0 [dB], the real operations for sorting columns of generator matrix are dominant for the (127, 64, 21) code. The values R_{RO} for both the (63, 30, 13) and the (127, 64, 21) code decrease as SNR becomes lower. This fact indicates that the proposed decoding algorithm becomes more efficient as the channel characteristics become worse. Furthermore, the value R_{RO} for the (127, 64, 21) code decreases faster than that for the (63, 30, 13) as SNR decreases.

For reference, we discuss the performance of trellis-based MLD algorithms [1], [2]. In Table II-8 of [1], for the (64, 30, 14) extended BCH code, Lafourcade and Vardy MLD algorithm [1] requires about $1.6 \cdot 10^7$ real operations. In Table II of [2], for the (64, 30, 14) extended BCH code, the upper bound of the real operations which RMLD-(G, U) algorithm requires is about $9.0 \cdot 10^6$. On the other hand, the proposed decoding algorithm requires $3.4 \cdot 10^4$ real operations on average at 1.5 [dB]. This implies the efficiency of the proposed decoding algorithm in some sense. In trellis-based MLD algorithms, however, the maximum number of real operations is bounded.

6. Concluding Remarks

In this paper, we have proposed a MRB based decoding algorithm for MLD. This algorithm is an improved version of the GS decoding algorithm, which reduces the time complexity for both constructing test error codewords and the number of real operations simultaneously. The theoretical analysis shows in total the efficiency of the proposed decoding algorithm, compared with the GS decoding algorithm. The results of computer simulation show the efficiency of the proposed decoding algorithm for the (63,30,13) BCH code and the (127,64,21) BCH code. Evidently, we can perform sub-optimal decoding by limiting the number of test error patterns. In that case, we can expect the same effect presented in this paper.

As future improvements, the extension to nonbinary cases is to be devised. Less stringent sufficient condition that eliminates unnecessary test error patterns is also to be derived.

Acknowledgement

The authors wish to thank the anonymous reviewers for their valuable and constructive comments.

This work is supported by Japan Society for the Promotion of Science under Grants-in-Aid for Scientific Research No. 1576-0281 and Waseda University Grant for Special Research Project No. 2001A-566.

References

[1] A. Lafourcade and A. Vardy, "Optimal sectionalization of a trellis," IEEE Trans. Inf. Theory, vol.42, no.3, pp.689–703, May 1996.
[2] T. Fujiwara, H. Yamamoto, T. Kasami, and S. Lin, "A trellis-based recursive maximum likelihood decoding algorithm for binary linear block codes," IEEE Trans. Inf. Theory, vol.44, no.2, pp.714–729, March 1998.
[3] Y.S. Han, C.P.R. Hartman, and C.C. Chen, "Efficient priority-first search maximum-likelihood soft decision decoding of linear block codes," IEEE Trans. Inf. Theory, vol.39, no.5, pp.1514–1523, Sept. 1993.
[4] D. Gazelle and J. Snyders, "Reliability-based code-search algorithm for maximum likelihood decoding of block codes," IEEE Trans. Inf. Theory, vol.43, no.1, pp.239–249, Jan. 1997.
[5] M.P.C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," IEEE Trans. Inf. Theory, vol.41, no.5, pp.1379–1396, Sept. 1995.
[6] M.P.C. Fossorier and S. Lin, "Reliability-based information set decoding of binary linear block codes," IEICE Trans. Fundamentals, vol.E82-A, no.10, pp.2034–2042, Oct. 1999.
[7] C.C. Shih, C.R. Wulff, C.R.P. Hartmann, and C.K. Mohan, "Efficient heuristic search algorithms for soft-decision decoding of linear block codes," IEEE Trans. Inf. Theory, vol.44, no.6, pp.3023–3038, Nov. 1998.
[8] T. Okada, M. Kobayashi, and S. Hirasawa, "An efficient heuristic search method for maximum likelihood decoding of linear block codes using dual codes," IEICE Trans. Fundamentals, vol.E85–A, no.2, pp.485–489, Feb. 2002.
[9] J. Snyders and Y. Be'ery, "Maximum likelihood soft decoding of binary block codes and decoders for the Golay codes," IEEE Trans. Inf. Theory, vol.35, no.5, pp.963–975, Sept. 1989.
[10] T. Koumoto, T. Kasami, and S. Lin, "A sufficient condition for ruling out some useless test error patterns in iterative decoding algorithms," IEICE Trans. Fundamentals, vol.E81–A, no.2, pp.321–326, Feb. 1998.
[11] T. Kasami, Y. Tang, T. Koumoto, and T. Fujiwara, "Sufficient conditions for ruling-out useless iteration steps in a class of iterative decoding algorithms," IEICE Trans. Fundamentals, vol.E82–A, no.10, pp.2061–2073, Oct. 1999.
[12] G.C. Clark, Jr. and J.B. Cain, Error-correction coding for digital communications, Plenum Press, NY, 1981.

Hideki Yagi was born in Yokohama, Japan, on Oct. 14, 1975. He received the B.E. degree and M.E. degree in Industrial and Management Systems Engineering from Waseda University, Tokyo, Japan, in 2001 and 2003, respectively. He is currently a doctorial student in Industrial and Management Systems Engineering at Graduate School of Waseda University. His research interests are coding and information theory.

Manabu Kobayashi was born in Yokohama, Japan, on Oct. 30, 1971. He received the B.E. degree, M.E. degree and Dr.E. degree in Industrial and Management Systems Engineering form Waseda University, Tokyo, Japan, in 1994, 1996 and 2000, respectively. From 1998 to 2001, he was a research associate in Industrial and Management Systems Engineering at Waseda University. He is currently a full-time lecturer of the Department of Information Science at Shonan Institute of Technology, Kanagawa, Japan. His research interests are coding and information theory and data mining. He is a member of the Society of Information Theory and Its Applications, Information Processing Society of Japan and IEEE.

Shigeichi Hirasawa was born in Kobe, Japan, on Oct. 2, 1938. He received the B.S. degree in mathematics and the B.E. degree in electrical communication engineering from Waseda University, Tokyo, Japan, in 1961 and 1963, respectively, and the Dr.E. degree in electrical communication engineering from Osaka University, Osaka, Japan, in 1975. From 1963 to 1981, he was with the Mitsubishi Electric Corporation, Hyogo, Japan. Since 1981, he has been a professor of School of Science and Engineering, Waseda University, Tokyo, Japan. In 1979, he was a Visiting Scholar in the Computer Science Department at the University of California, Los Angels (CSD, UCLA), CA. He was a Visiting Researcher at the Hungarian Academy of Science, Hungary, in 1985, and at the University of Trieste, Italy, in 1986. In 2002, he was also a Visiting Faculty at CSD, UCLA. From 1987 to 1989, he was the Chairman of Technical Group on Information Theory of IEICE. He received the 1993 Achievement Award, and the 1993 Kobayashi-Memorial Achievement Award from IEICE. In 1996, he was the President of the Society of Information Theory and Its Applications (Soc. of ITA). His research interests are information theory and its applications, and information processing systems. He is an IEEE Fellow, and a member of Soc. of ITA, the Operations Research Society of Japan, the Information Processing Society of Japan, the Japan Industrial Management Association, and Informs.