Memoirs
of the National Defense Academy,
Japan, Vol. 62, No. 2,
(2022), pp. 11– 20

< Paper >

Graph-theoretic Code on Grid Graph under Communication with High Noise

By Masaki FUKUSHIMA* and Seiji KATAOKA**

Abstract

Graph-theoretic code is a classical form of error-correcting code. In graph-theoretic code, each codeword consists of a group of cycles in the graph. This structure is suitable for applying mathematical programming techniques that attain maximal likelihood decoding. Though the general evaluation of graph-theoretic code, in practical viewpoint, is not so high, if graph-theoretic code is applied on grid graphs, it demonstrates excellent performance under communication with high noise. Grid graphs are easily implemented on computers and naturally grasp information as images. These characteristics make it possible to develop new encoding methods and to improve error-correcting ability. Our proposed method comes true best-seen error-correcting results under communication with high noise.

Keywords: Graph-theoretic code, Mathematical programming decoding, Grid graph, Communication with high noise

1. Introduction

Graph-theoretic code is a classical form of error-correcting code that has been theoretically studied from various viewpoints.[4–6] Graph-theoretic code is beneficial in that each codeword corresponds to a cycle group in the graph. This feature makes it suited to mathematical programming, particularly graph-network programming, where various optimization problems and their efficient algorithms have been studied.[1,7] Hence, graph-theoretic code is one of rare codes that polynomially attain maximal likelihood (ML) decoding.[6] Furthermore, state-of-the-art practical solvers such as Gurobi[3] also allow graph-theoretic code to attain high-speed ML decoding.

In terms of practicality, however, graph-theoretic code has been poorly evaluated because it is simply a restricted version of low density parity check (LDPC) code[8] and its decoding ability is not particularly high.[5] However, applying graph-theoretic code on grid graphs leads to far better performance than that by LDPC code, especially when the signal-to-noise ratio is low, that is, under communication with high noise.

Grid graphs are structurally simple and have various practical applications. Furthermore, grid graphs are convenient not only to programmatically implement, but also to visually grasp information. These characteristics allow developing new encoding methods and proposing strategies for improving error-correcting abilities.

* Undergraduate Student(63rd term)
** Professor

The remainder of this paper is organized as follows: Section 2 introduces graph-theoretic code and ML decoding formulation. Computational experiments show the superiority of using simple formulation and solver, and show the high decoding performance on grid graphs under communication with high noise. In Section 3, we show a method for embedding block code like long vectors into the two-dimensional plane of a grid graph. This method also leads to an easy way of making a codeword directly from information bits, namely, a way to form a cycle group in a grid graph. The proposed coding method is applicable to any type of information bits, but applying it to binary images will help us understand its effectiveness. The visualized results both demonstrate the superiority of our proposed method and reveal some defects. In Section 4, we develop an improvement strategy to eliminate these defects. Computational experiments demonstrate best-seen error-correcting results under communication with high noise. Our concluding remarks are in Section 5.

2. Graph-theoretic code and decoding by mathematical programming

2.1 Graph-theoretic code

Consider an undirected graph (Figure 1) and a spanning tree in the graph (bold lines). The set of edges not in the spanning tree is called the *co-tree*. In Figure 1, for example, the co-tree is $\{e_1, e_2, e_3\}$.

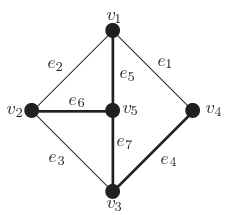

Fig. 1 Undirected graph with 5 nodes and 7 edges. The bold lines show a tree.

Adding a co-tree edge to the spanning tree creates a unique cycle. The relation between each co-tree edge and its corresponding cycle defines a 0–1 matrix, where rows are the co-tree edges and columns are all edges in the graph. This matrix is called the *fundamental circuit matrix*. Graph-theoretic code uses the fundamental circuit matrix as a generating matrix. Note that the information bits correspond to co-tree edges, so let k be the length of information bits and n be the length of codeword. It is known that any group of cycles is generated as the exclusive OR (XOR) of appropriate rows of the fundamental circuit matrix.[11] For example, the cycle $(e_1, e_5, e_6, e_3, e_4)$ is generated by the XOR of cycle (e_1, e_5, e_7, e_4) by e_1 and cycle (e_3, e_7, e_6) by e_3. Therefore, each codeword also corresponds to a cycle group in the graph.

Deleting an edge from the spanning tree divides the set of vertices into two groups. The set of edges between the two separated groups is called a *cut*. The relation between each edge in the spanning tree and its corresponding cut defines a 0–1 matrix, where rows are edges in the spanning tree and columns are all edges in the graph. This matrix is called the *fundamental cut matrix*. It is known that the fundamental cut matrix is the dual matrix of the fundamental circuit matrix, so we can use the fundamental cut matrix as a parity check matrix for graph-theoretic code.

It is also known that any cut can be generated as the XOR of appropriate rows of the fundamental cut matrix.[11] For example, the cut $\{e_4, e_7, e_3\}$ is generated by the XOR of cut $\{e_1, e_4\}$ by e_4 and cut $\{e_1, e_7, e_3\}$ by e_7. Then, every cut between a vertex and the other vertices, that is, the *vertex–edge incidence matrix*, is also constructed from the fundamental cut matrix. Therefore, in this paper, we use the vertex–edge incidence matrix H as the parity check matrix.

2.2 Maximal likelihood decoding by mathematical programming

This study assumes binary phase-shift keying (BPSK) and additive white Gaussian noise (AWGN) as the communication channel, which are normally used in studies of satellite or wireless communications. That is, for a codeword $\boldsymbol{w}$, the corresponding sent word is formed by transforming 0 into 1, and 1 into -1. While conveying the sent word through the channel, a value following a Gaussian distribution with mean 0.0 and variance σ^2 is independently added as noise to each bit. Here, $\sigma^2 = 10^{-A/10}/(2R)$, where A [dB] is the signal-to-noise ratio (SNR) and code rate R = (the information length) / (the codeword length) = k/n. The result becomes the received word $\boldsymbol{r}$.

At this time, the conditional probability of receiving r_j when sending w_j is

$$P_{\boldsymbol{r}|\boldsymbol{w}}(r_j|w_j) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_j-(1-2w_j))^2}{2\sigma^2}\right). \tag{1}$$

Additionally, consider the logarithm likelihood ratio

$$\lambda_j := \log\left(\frac{P_{\boldsymbol{r}|\boldsymbol{w}}(r_j|0)}{P_{\boldsymbol{r}|\boldsymbol{w}}(r_j|1)}\right). \tag{2}$$

Maximizing the probability $\prod_j P_{\boldsymbol{r}|\boldsymbol{w}}(r_j|w_j)$ is equivalent to minimizing the following linear objective function[2] subject to constraints for the codewords on Galois field 2 (GF(2)):

$$\begin{aligned} \min\quad & \boldsymbol{\lambda}^\top \boldsymbol{x}, \\ \text{s.t.}\quad & H\boldsymbol{x} \equiv \boldsymbol{0} \pmod 2, \\ & \boldsymbol{x} \in \{0,1\}^n. \end{aligned} \tag{3}$$

The optimal 0–1 solution $\boldsymbol{x}^*$ to (3) leads the ML decoding.

In order to solve the formulation (3) on GF(2), we simply reformulate the formulation on real field such as (4), where $H\boldsymbol{x}$ is a vector of degrees on each vertex, and $-2\boldsymbol{y}$ is introduced to force every degree to be even:

$$\begin{aligned} \min\quad & \boldsymbol{\lambda}^\top \boldsymbol{x}, \\ \text{s.t.}\quad & H\boldsymbol{x} - 2\boldsymbol{y} = \boldsymbol{0}, \\ & \boldsymbol{x} \in \{0,1\}^n, \boldsymbol{y} : \text{nonnegative integer}. \end{aligned} \tag{4}$$

Then, a mathematical programming solver such as Gurobi can be applicable to obtain $\boldsymbol{x}^*$. We call this decoding strategy *mathematical programming decoding*.

As introduced in Section 1, for graph-theoretic code, Kashyap[6] proposed a polynomial time ML decoding algorithm. The algorithm reduces the method to the Euler subgraph problem and obtains the optimal set of edges called T-join. Furthermore, obtaining optimal T-join consists of a number of shortest path problems and the minimum-weight perfect matching problem for a general complete graph.[7] Indeed, it is polynomial but the order is high. We wrote a code of the algorithm but observed that, in practice, the performance is inferior to that of the mathematical programming decoding.

2.3 Performance of graph-theoretic code

Table 1 shows computational results of graph-theoretic code on random graphs and grid graphs to evaluate bit-error rate (BitE)[%] and cpu runtime (CPU)[sec]. For comparison, we also show results for an LDPC(n,k) code referred from MacKay,[9] where n is code length and k is information bits. As the decoding algorithm, we use Feldman's formulation[2] with 0–1 conditions for ML decoding (5), where $\boldsymbol{h}_i$ is the set of nonzero-column subscripts in the ith row of the parity check matrix H.

$$\begin{aligned} \min\quad & \boldsymbol{\lambda}^\top \boldsymbol{x}, \\ \text{s.t.}\quad & \textstyle\sum_{j\in S} x_j - \sum_{j\in \boldsymbol{h}_i \setminus S} x_j \le |S| - 1, \\ & \quad \forall i, S \subseteq \boldsymbol{h}_i, |S| \text{ is an odd integer}, \\ & \boldsymbol{x} \in \{0,1\}^n. \end{aligned} \tag{5}$$

Table 1 The computational results of random graphs and grid graphs.

	LDPC(204,102)		RndG(200,100)		RndG(400,200)		GridG(200,101)		GridG(450,226)		GridG(800,401)	
SNR	BitE	CPU	BitE	CPU	BitE	CPU	BitE	CPU	BitE	CPU	BitE	CPU
0.0	11.361	41.469	10.328	0.024	10.212	0.048	9.109	0.028	9.156	0.083	9.083	0.280
0.5	6.360	25.671	8.253	0.023	8.250	0.046	6.595	0.024	6.562	0.074	6.441	0.213
1.0	2.259	11.840	6.220	0.022	6.278	0.044	4.465	0.021	4.304	0.063	4.228	0.144
1.5	0.475	3.581	4.212	0.020	4.384	0.042	2.793	0.017	2.751	0.049	2.682	0.103
2.0	0.052	0.885	2.652	0.017	2.732	0.035	1.665	0.015	1.735	0.039	1.692	0.078
2.5	0.000	0.130	1.569	0.015	1.481	0.030	1.003	0.013	1.048	0.031	0.985	0.062
3.0	0.000	0.054	0.894	0.012	0.623	0.025	0.543	0.012	0.604	0.026	0.608	0.051
3.5	0.000	0.048	0.415	0.011	0.301	0.020	0.286	0.011	0.300	0.021	0.346	0.042
4.0	0.000	0.047	0.196	0.009	0.125	0.017	0.144	0.010	0.166	0.187	0.182	0.036
4.5	0.000	0.048	0.086	0.009	0.044	0.014	0.061	0.009	0.093	0.016	0.083	0.031
5.0	0.000	0.047	0.042	0.008	0.022	0.013	0.022	0.008	0.039	0.015	0.039	0.028

Numerical experiments described in this paper are performed on a personal computer with a 3.3 GHz Intel Core i7-5820K CPU running the CentOS 6.8 operating system and Gurobi 9.0[3] as the mathematical programming solver. Each figure shows average values for 1000 trials.

As for random graph, RndG(n, k) has $n-k+1$ vertices, $n-k$ randomly selected spanning tree edges, and additional k randomly selected co-tree edges. And as for grid graph, GridG(n, k) has $\sqrt{n/2}\times\sqrt{n/2}$ grids and $k = n/2+1$ co-tree edges.

Observing Table 1 under middle- or low-noise communications, we can see that BitE of LDPC code is much smaller than that of graph-theoretic code. But under high-noise conditions, graph-theoretic code shows superior decoding performance. Furthermore, mathematical programming decoding is ML, and this decoding strategy can certainly obtain a codeword.

Moreover grid graphs lead to decoding performance superior to random graphs, particularly when SNR is small. When SNR is 0.0 [dB], the bit-error rate is less than 10%. Computational performance is fast even when the code length becomes large, as in GridG$(800, 401)$.

Graph-theoretic code has generally been poorly evaluated for decoding performance, and we saw similar phenomena when using random graphs. But when using a grid graph, the graph-theoretic code leads to preferable decoding performance, especially when SNR is small, i.e., under communications with high noise.

3. Coding strategy and performance

In this section, we propose a coding strategy that provides a meaningful connection between information bits and cycle groups in grid graphs. In the proposed strategy, besides the conventionally used bit error, we introduce another error evaluation criterion called *pixel error*.

3.1 Coding strategy on grid graph

As an example, consider a grid graph having 3×4 squares, like that on the left side in Figure 2. This graph has $(3+1)(4+1) = 20$ vertices and $3(4+1)+4(3+1) = 31$ edges. The bold edges show a spanning tree and the remaining $31-(20-1) = 12$ edges (co-tree edges) are numbered $e_1, e_2, \ldots, e_{12}$. Since co-tree edges correspond to information bits, an information bit sequence such as $(0,1,0,1,1,1,1,0,0,0,1,0)$ induces six cycles, and their XOR produces the cycle shown on the right side in Figure 2.

Since the length of information bits is equal to

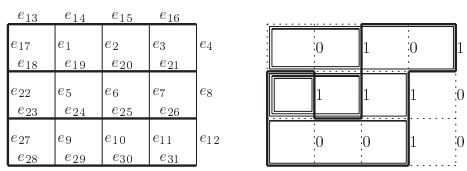

Fig. 2 Conventional coding method.

the number of squares in the grid graph, we consider assigning information bits to squares in the grid graph such as the black dots in Figure 3. The corresponding codeword is then set to the group of cycles enclosing the dots in the squares. See the bold edges in Figure 3.

This codeword coincides with the result for information bits such as $(1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0)$ in the conventional coding method, but the proposed coding method visually reflects the information bits as a binary image. This characteristic plays an important role in precise error correcting. Section 4 shows a strategy for improving error correction based on this characteristic.

Additionally, the proposed coding method induces an easy algorithm for constructing codewords, namely, tracing the outlines of ON pixels (black dots) in squares in the grid graph. This algorithm sets an edge to 1 when two pixels on both sides of the edge are ON–OFF or OFF–ON, or to 0 when the two pixels are ON–ON or OFF–OFF. Note that the outside area is regarded to be OFF. This algorithm does not need to take a spanning tree, make cycles for each co-tree edge, or calculate the XOR for those cycles. Instead, it can be performed directly from the pixel states. The reverse procedure, getting information bits from cycle groups, is also easy; we simply alternatively switch ON and OFF each time we step over the edges of cycles.

3.2 Pixel error

Bit error is a standard criterion for evaluating the performance of error-correcting code. For example, if Figure 4 is the decoded result for an original codeword like that in Figure 3, then bit errors occur at the eight edges marked $\times$, and the bit-error rate is $8/31 \approx 0.258$. In the conventional coding method, information bits correspond to co-tree edges $e_1, \ldots, e_{12}$, so the three information bits marked as $\otimes$ edges are incorrectly conveyed, and the bit-error rate for that information part is $3/12 = 0.250$. In the proposed coding method, however, information bits correspond to squares in the grid graph, which we called *pixels* in the previous subsection. Hence, the two pixels marked $\odot$ are errors in the information part, so the error rate is $2/12 \approx 0.167$. Below, we call this new evaluation criterion *pixel error*.

This example shows that two error rates concerning information bits can be obtained from the same decoded result. The aim of error-correcting code is to convey information bits as correctly as possible. Hence, it should be meaningful to pay attention to error in information bits.

3.3 Decoding performance of the two coding methods

In section 3.1, we have introduced two coding methods for assigning information bits for graph-theoretic code on a grid graph. One is a conventional method that assigns information bits to co-tree edges, and the other is the proposed method, which assigns them to squares in a grid graph. This subsection examines the differences through computational experiments.

Before showing those results, we present two observed results from several preliminary exper-

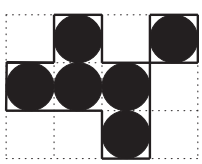

Fig. 3 Proposed coding method.

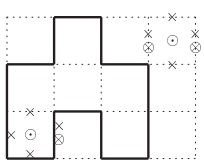

Fig. 4 Pixel error.

iments. The first result is related to spanning trees. Testing various spanning trees, we determined that no conspicuous influences among these trees were observed. The second result is related to types of binary images. We tested several binary images, including a designed logotype like that in Figure 5 (the OR logo at the top center), Japanese kanji characters, several block-pixel squares arrayed like a chessboard, and QR codes as random information bits. In these preliminary experiments, we observed no significant differences among these binary images. Hence, in the following experiments, results are for the logotype image in Figure 5, which consists of $27 \times 25 = 675$ pixels (information bits) and 1402 edges (code length).

In Table 2, 'Conventional' shows the results of the conventional coding method, which does not reflect the shape of the binary image. 'Proposed' shows the results of the proposed coding method, which outlines the shape of the binary image. 'BitE' is the average bit-error rate for the codeword. 'InfE' (Conventional) and 'PxlE' (Proposed) are average error rates for information bits. Each figure in the table is an averaged value from 1000 trials. Figures in parentheses are standard deviations '(sd)' for the error rate.

Table 2 shows that both BitE and InfE take similar values in the conventional coding method.

Table 2 Comparison of the conventional and proposed coding methods.

	Conventional			Proposed		
SNR	BitE (sd)	InfE (sd)	CPU	BitE (sd)	PxlE (sd)	CPU
0.0	8.692(2.2)	8.742(2.3)	1.078	8.711(2.2)	10.248(6.3)	1.030
0.5	6.256(1.8)	6.302(2.0)	0.557	6.260(1.8)	6.042(3.8)	0.532
1.0	4.325(1.5)	4.347(1.6)	0.299	4.383(1.5)	3.649(2.3)	0.277
1.5	2.896(1.1)	2.914(1.2)	0.166	2.896(1.1)	2.116(1.3)	0.166
2.0	1.880(0.9)	1.891(0.9)	0.108	1.869(0.9)	1.248(0.8)	0.111
2.5	1.188(0.7)	1.194(0.7)	0.076	1.169(0.7)	0.735(0.5)	0.078
3.0	0.694(0.5)	0.697(0.5)	0.057	0.697(0.5)	0.419(0.3)	0.059
3.5	0.404(0.4)	0.409(0.4)	0.046	0.386(0.4)	0.223(0.2)	0.049
4.0	0.219(0.3)	0.223(0.3)	0.039	0.211(0.3)	0.119(0.2)	0.041
4.5	0.109(0.2)	0.111(0.2)	0.036	0.105(0.2)	0.056(0.1)	0.033
5.0	0.052(0.1)	0.053(0.1)	0.035	0.046(0.1)	0.024(0.1)	0.028

This is because the conventional coding method does not consider the shape of the original binary image. A similar occurrence is also observed for BitE under the proposed coding method.

In contrast, PxlE for the proposed coding method takes approximately half the InfE values seen for the conventional coding method under middle- or low-noise communications. This implies that the proposed coding method certainly makes use of the shapes of binary images. However, as Table 2 shows, the average pixel-error rate is around 10% at 0.0 [dB]. This result may not seem good at a glance, but the standard deviation value too becomes large. This implies that even if the decoding results have the same bit errors, they could have very different pixel errors. Section 3.4 shows examples from a binary image and a raw photo image. These visual results will lead to an algorithm that improves decoding accuracy.

3.4 Decoding results for binary and raw photo images

First, we present two decoded results from 1000 trials at SNR 0.0 [dB] in Table 2. The chosen results have the same bit errors but different pixel errors. In Figure 5, the top panel (a) is the original codeword, the leftmost panels ((b) and (e)) are the results of hard decisions, where we take edge e if r_e is negative, and do not take e otherwise. The center panels ((c) and (f)) show the results of a sum–product method[8] applied to graph-theoretic code. The sum–product method is an approximate algorithm that reduces errors as Humming distance, so the results usually will not construct feasible solutions (cycle groups).

The rightmost panels ((d) and (g)) are the results of decoding by mathematical programming, so these results form cycle groups. Note that both chosen results have same numbers of bit errors 116, but different numbers of pixel errors, 67

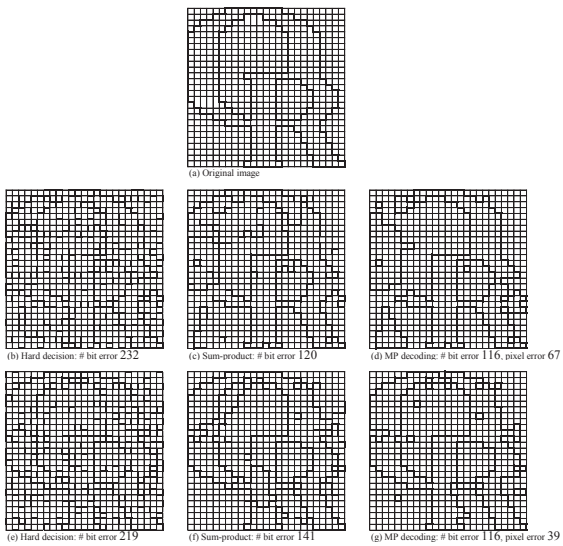

Fig. 5 Binary images, where (d) and (g) have the same 116 bit-errors.

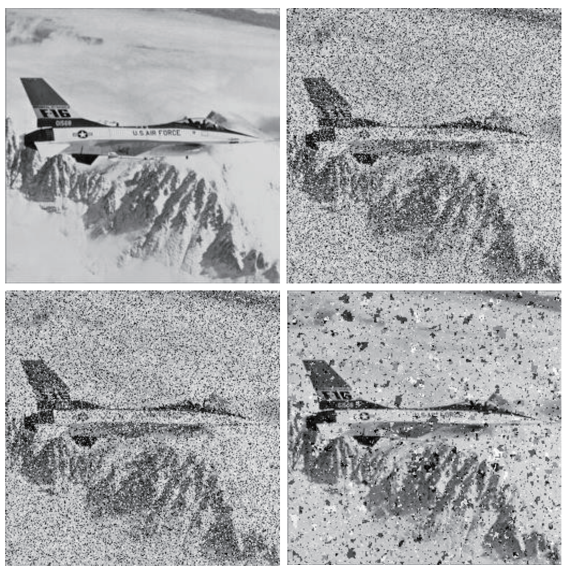

Fig. 6 Raw photo images. Upper left: original; upper right: hard; lower left: sum-product; lower right: proposed method.

and 39, respectively. This wide deviation in pixel errors is also observed in Table 2 when SNR is small. In such cases, some erroneously connected cycles or somewhat larger cycles in impossible areas are observed, such as the larger cycle in the bottom-left corner in Figure 5(d), which includes 20 pixels.

We next used a raw photo image from the Standard Image Database.[10] This raw photo image consists of 256×256 pixels with 256 gradations. That is, eight 2^g-gradation ($g = 0, 1, \ldots, 7$) binary images, each having 256×256 pixels, are stacked.

Figure 6 shows the decoded results of the photo image for 0.0 [dB] AWGN. The upper-left image is the original photo, and those at the upper right and lower left are images resulting from the hard-decision and sum–product methods, respectively. Since codewords of graph-theoretic code consist of graph edges, and noise is added to edges, decoded results do not form cycle groups. See also Figure 5(b), (c), (e), and (f). Then it is impossible to set each grid-graph pixel to ON or OFF. Hence, the hard-decision and sum–product results are from the conventional method.

The lower-right image in Figure 6 is the result of graph-theoretic code and decoding by mathematical programming. Noise is reduced to the extent that marks and characters on the airplane become readable. However, somewhat larger fragmental noise is also observed, similar to that in Figure 5(d). The following section presents a strategy for eliminating these large erroneous cycles or fragmental noises by making use of the grid-graph structure and information around each edge.

4. Improving strategy

4.1 Ethane graph

For a given grid graph, consider a subgraph consisting of an edge e, vertices u, v on either side

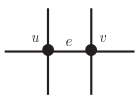

Fig. 7 Ethane graph G_E.

of e, and six edges connecting to u and v (Figure 7). We call this subgraph an *ethane graph* G_E, because it is similar to the molecular structure of ethane (C_2H_6).

In the ethane graph G_E, possible edges or degrees are strongly restricted in that each codeword consists of a cycle group. First of all, the degrees of u and v must be even, so the total number of edges in G_E, denoted $|E(G_E)|$, cannot be 1, 5, or 6. Secondly, for example when $|E(G_E)| = 2$, the degrees of u and v must be 2–0 or 0–2, and the center edge e must not be taken. In this case, the number of edge patterns in G_E is limited to six: those at either side of u and v and the two non-center edges. Considering BPSK, in which 1 is sent as -1 and 0 is sent as 1, Table 3 summarizes the expected received values of edges in G_E, where $R(e)$ is the expected received value for center edge e and $R(G_E)$ is the total amount of expected received values for the edges in $E(G_E)$.

4.2 Revising direction to reverse added noise

The ethane graph and possible states in Table 3 are used to appropriately revise received values. We show the procedure along with an example. In Figure 8, the number beside each edge is the received value r_j. For an ethane graph of bold edges, we calculate the total amount of received value r_j ($j \in E(G_E)$). In this example, this is

Table 3 Possible states in G_E.

$\lvert E(G_E)\rvert$	0	1	2	3	4	5	6	7
$\deg(u, v)$	0-0	—	2-0,0-2	2-2	2-2	—	—	4-4
$R(e)$	1	—	1	−1	1	—	—	−1
$R(G_E)$	7	—	3	1	−1	—	—	−7

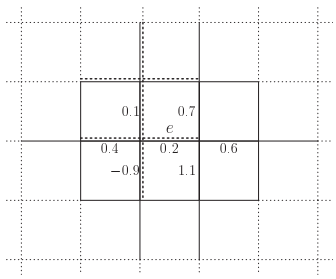

Fig. 8 Received values on the ethane graph of bold edges.

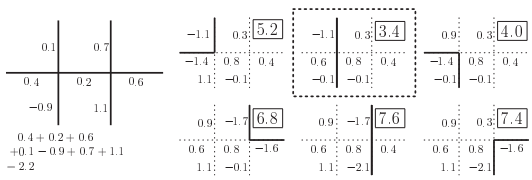

Fig. 9 Possible edge patterns in the $R(G_E) = 3$ case.

$0.4 + 0.2 + 0.6 + 0.1 - 0.9 + 0.7 + 1.1 = 2.2$. According to Table 3, the nearest total amount of expected received value $R(G_E)$ is 3. From this information, we estimate an edge pattern in the ethane graph as described below.

$R(G_E) = 3$ indicates that two edges lie in G_E, the degrees of vertices are 2–0 or 0–2, and the center edge is not used. In this case, six edge patterns are possible (Figure 9). For each possible edge pattern, we calculate the reversing values of added noises by AWGN. In the case of an edge with $r_j = 0.4$, if the edge is taken as part of a cycle, the reversing value of added noise is -1.4, or 0.6 if the edge is not taken as part of the cycle. In Figure 9, the number beside each edge is the reversing value of added noise for the possible edge patterns. Additionally, the values in boxes for each edge pattern show sums of absolute values of reversing added noise. In this case, 3.4 is the minimum among the six edge patterns. Therefore, it is natural to estimate that the edge

pattern in the dotted box fits part of the original codeword through the ethane graph. Let $\delta_1(j)$ ($j \in E(G_E)$) be the revising direction to reverse added noise obtained from the bold ethane graph in Figure 8.

The first procedural step is to calculate the total amount of received values and to find the nearest expected received value $R(G_E)$ from Table 3. The second estimating step is executed based on this information. If the guess in the first step is wrong, the resultant $\delta_1(j)$ ($j \in E(G_E)$) will also give the wrong revising direction. However, considering several ethane graphs will mitigate serious mistakes. For edge e in Figure 8, the bold-dotted ethane graph is the second ethane graph, so $\delta_2(e)$ is induced as another revising direction. Observing Figure 8, at most seven ethane graphs include edge e, and these ethane graphs cover the solid edges in the figure. Considering information on the broadly surrounding areas,

$$\Delta(e) := \delta_1(e) + \delta_2(e) + \cdots + \delta_7(e) \qquad (6)$$

can be expected to give a revised direction, and $\Delta(e)$ is used to revise the received value r_e to r'_e,

$$r'_e := r_e + \theta\Delta(e), \quad \forall\ e \text{ in grid graph.} \qquad (7)$$

Through various preliminary experiments, we ensured that an appropriate step size θ depends on the noise strength, that is, SNR. However, the value of SNR cannot be known in advance, so instead we use the rate of odd-degree vertices in hard decisions and find an appropriate step size θ as follows:

$$\theta := 0.21 \cdot (\text{the rate of odd-degree vertices}). \qquad (8)$$

4.3 Effect of improving strategy

To see the effects of the improving strategy, we use the same experimental data as in Section 3. As the first results, Table 4 shows improvements over the results shown in Table 2. The left half of this table is a copy of Table 2, and the right half shows the results of the improving strategy.

These results clearly demonstrate that the improving strategy is effective. Particularly under communication with high noise, this strategy reduces error to less than half that of both BitE and PxlE. Furthermore, cpu runtimes are decreased. As Table 2 shows, decoding by mathematical programming performs quickly as added noise is lowered. The revised received word is therefore regarded as having reduced noise as a welcomed side-effect.

The second result is Figure 10, which shows the improvements of Figures 5 and 6. The left half shows improvement to the binary images in Figure 5. The left images ((a) and (c)) are copies of the results of the original mathematical programming decoding (Figure 5(d) and (g)), and the right images ((b) and (d)) are the improved results. In this case, too, we can see that somewhat larger cycles are eliminated. The right half shows the result of the raw photo image in Figure 6, in which we can also see that most large fragmental noise is eliminated, and characters on the airplane can be clearly recognized.

5. Concluding remarks

Graph-theoretic code is a classical form of error-correcting code, but it has generally not

Table 4 Numerical results of the improvement.

	Proposed			Improved		
SNR	BitE(sd)	PxlE(sd)	CPU	BitE(sd)	PxlE(sd)	CPU
0.0	8.711(2.2)	10.248(6.3)	1.030	4.321(1.5)	3.881(3.3)	0.418
0.5	6.260(1.8)	6.042(3.8)	0.532	3.167(1.2)	2.438(1.8)	0.215
1.0	4.383(1.5)	3.649(2.3)	0.277	2.272(1.0)	1.599(1.0)	0.139
1.5	2.896(1.1)	2.116(1.3)	0.166	1.572(0.8)	1.036(0.7)	0.100
2.0	1.869(0.9)	1.248(0.8)	0.111	1.071(0.6)	0.672(0.5)	0.076
2.5	1.169(0.7)	0.735(0.5)	0.078	0.694(0.5)	0.418(0.3)	0.058
3.0	0.697(0.5)	0.419(0.3)	0.059	0.430(0.4)	0.248(0.2)	0.049
3.5	0.386(0.4)	0.223(0.2)	0.049	0.253(0.3)	0.143(0.2)	0.042
4.0	0.211(0.3)	0.119(0.2)	0.041	0.145(0.2)	0.080(0.1)	0.035
4.5	0.105(0.2)	0.056(0.1)	0.033	0.080(0.2)	0.043(0.1)	0.030
5.0	0.046(0.1)	0.024(0.1)	0.028	0.035(0.1)	0.018(0.1)	0.026

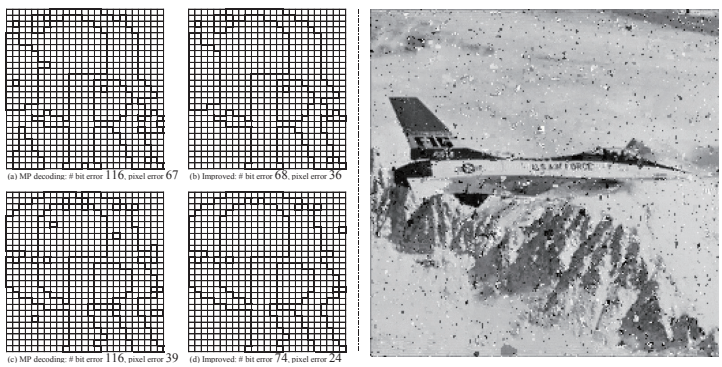

Fig. 10 Improved results for binary and raw photo images.

been well evaluated. In Section 2, however, we show that graph-theoretic code using grid graphs and decoding by mathematical programming works well, particularly under communication with high noise. Furthermore, Sections 3 and 4 effectively advance this feature to practical use. To remove the inferiority, we introduced ethane graphs, which strongly limit the number of edges, degrees, and edge patterns. This limitation allows reversal of added noises, eliminating the inferiority by revising received words.

When received words are sent through space or in deliberately jammed communications, it might be impossible to ask the sender to resend messages or to use longer codewords. In such high-noise cases, the proposed strategy might be practically useable method.

References

1) R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall (1993).
2) J. Feldman, M.J. Wainwright and D. Karger, "Using linear programming to decoding binary linear codes," *IEEE Transaction Information Theory*, **51** (2005), pp.954-972.
3) Gurobi Optimizer. http://www.gurobi.com Accessed 30 August 2022.
4) S.L. Hakimi and J.G. Bredeson, "Graph theoretic error-correcting codes," *IEEE Transaction Information Theory*, **IT-14** (1968), pp.584-591.
5) X.Y. Hu and E. Eleftheriou, "Binary representation of cycle Tanner-Graph GF(2b) codes," *IEEE International Conference on Communications*, **1** (2004), pp.528-532.
6) N. Kashyap, "A decomposition theory for binary linear codes," *IEEE Transaction on Information Theory*, **54** (2008), pp.3035-3058.
7) B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, Springer (2000).
8) S. Lin and D.J. Costello, *Error Control Coding (2nd ed.)*, Prentice Hall (2004).
9) D.J.C. MacKay, *Encyclopedia of Sparse Graph Codes*, http://www.inference.phy.cam.ac.uk/mackay/ Accessed 30 August 2022.
10) Standard Image Data-Base, http://www.ess.ic.kanagawa-it.ac.jp/app_images_j.html Accessed 30 August 2022.
11) Y. Takenaka, *Liniar Alegibraic Graph Theory*, Baifuukan (1989). (in Japanese).