信州大学
大学院総合理工学研究科

修士論文

誤り訂正符号としてのラテン方陣の復号方法について

指導教員　西新 幹彦 准教授

専攻　工学専攻
分野　電子情報システム工学分野
学籍番号　19W2077A
氏名　塚田 芳寿

2021 年 3 月 22 日

目次

1　序論

1.1　研究背景

現代社会で情報の伝送はスマートフォンやパソコンなど身の回りの様々なところで行われている．シャノンは現実の通信の流れを図 1 のようにモデル化した [1]．送信者の送ったメッセージは，符号器によって符号化され通信路を通る．通信路から出てきた受信語は，復号器によって復号され受信者に伝わる．通信路で雑音の影響を受ける場合，受信者は確実に正しいメッセージを受け取ることはできないが，通信はできるだけ高確率で正しい情報を受けとれるようにしたいという確率的問題として考えることができる．また，図 1 のように符号化は情報源符号化と通信路符号化に分けられる．誤り訂正符号とは，通信路符号化によって送る情報を工夫し，送られてきた情報を復号器で可能な限り正しく復号する規則を考えることによって，通信の質を向上させる技術である．情報の符号化の方法は一通りではなく，さまざまな工夫の仕方が考えられる．過去の研究では，数独というものに着目した研究がある [2]．

数独はパズルの 1 種である．数独の厳密なルールは文献 [3] に記載がある．数独では，いくつか空白がある表をヒントに，元の表がなんであったかを推測する．この手順は受信した情報から，送られた情報がなんであったかを当てる手順と同じである．このように数独は誤り訂正符号の 1 種であると考えることができる．

本研究ではラテン方陣を誤り訂正符号として着目し考察を行う．ラテン方陣は数独と似ており，数独からいくつかのルールを取り除いたものがラテン方陣になる．したがって，ラテン方陣も数独と同様に誤り訂正符号と考えることができ，送信者はラテン方陣を送り，受信者は受け取った表を見て元のラテン方陣を推測することができる．過去に行われたラテン方陣の研究では，数独と同様に空白を埋めるという問題設定であった [4]．本研究では空白ではなく，誤った数字が表に入っているという問題設定とする．

符号語としてラテン方陣のルールを採用するため，本研究で手を加えることができる箇所は復号化の方法ということになる．ただし，過去に行われた数独に関する研究で，数独を誤り訂正符号として用いる場合，通信に実用化するには難しい性能であることがわかっている．数独と似ているラテン方陣も同様に，そのまま誤り訂正符号としての実用化は困難であると考えている．しかし，ラテン方陣を誤り訂正符号とし，最適な復号手法を考案することで，その考え

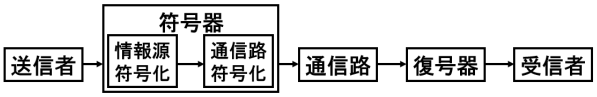


図 1　シャノンによる通信のモデル

方や手法が他の誤り訂正符号の復号方法に応用できないかと考えている．したがって，本研究ではラテン方陣のルールで符号化することによる性能の良し悪しについては議論せず，復号手法によって性能を向上させることについてのみ議論する．本研究では復号方法として，実用化されている sum-product 復号について議論する．sum-product は計算を高速に行うことができる．しかし，sum-product は万能ではなく，符号の構造によっては復号性能が悪化してしまう場合がある．本研究では，ラテン方陣の特殊な符号の構造に着目し，sum-product によって生じる問題を解決する方法を模索する．

1.2　本論文の構成

1 章で本研究の背景，目的を述べた．2 章で一般的な誤り訂正符号の復号方法の数学的な原理について述べる．3 章で実際にラテン方陣を誤り訂正符号としたときの問題設定，既存の復号方法による復号について述べる．4 章でラテン方陣の復号方法の提案手法とシミュレーション結果，評価について述べる．5 章では 4 章とは別の提案手法について，概要とシミュレーション結果，評価を述べる．6 章では 4 章と 5 章の結果を踏まえたうえで，sum-product アルゴリズムの計算の繰り返しについて述べる．7 章では本研究の結論と今後の展望について述べる．

2　誤り訂正符号の復号方法の原理

本章では文献 [5] に基づき，一般的な誤り訂正符号の原理，復号方法の原理について述べる．

2.1　誤り訂正符号の概要

図 2 のように，通信路として最もシンプルな 2 元対称通信路について考える．送りたい情報は 0, 1 の 2 つであり，確率 p で異なる情報に変わってしまう．

最も簡単な誤り訂正符号として繰り返し符号がある．繰り返し符号の手順を図 3 に示す．送

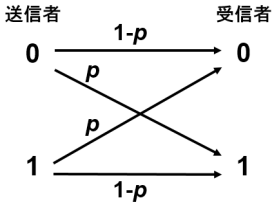


図 2　2 元対称通信路

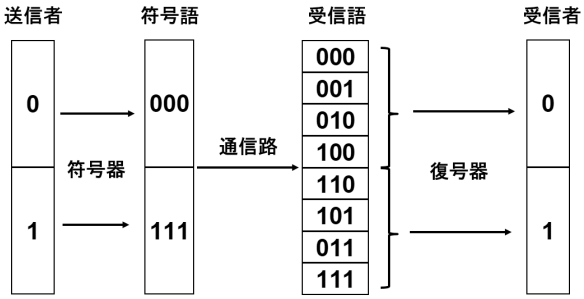


図 3　3 回繰り返し符号

信者は送りたい情報を 3 回繰り返して送る．受信者は受け取った 3 つの数字を見て 0, 1 の数が多いほうが送られてきたと判定する．この時，0 が送られてきたときに受信者が 0 と判定するのは受信語が 000, 001, 010, 100 の時の 4 パターンである．0 を送った時に 000 が起こる確率は $(1-p)^3$, 001, 010, 100 が起こる確率は $p(1-p)^2$ である．よって 0 を送った時に受信者が 0 と判定する確率はそれぞれの確率の和をとって

$$(1-p)^3 + 3 \times p(1-p)^2 \tag{1}$$

となる．$p = 0.1$ だった場合，そのまま送った場合に 0 を正しく復号できる確率は 0.9 であるが，繰り返し符号を使った場合，式 (1) に $p = 0.1$ を代入すると，0 を復号できる確率は 0.972 となり，そのまま送るよりも正しく送られる確率を上げることができる．繰り返し符号では送る情報を 3 回繰り返して送ることで復号誤り確率を良くすることができる．しかし，正しく送られる確率を上げるために情報を 3 回送っているため，そのまま送るよりも 2 回分余分に情報を送るという手間をかけていることになる．できるだけ情報を正しく送りつつ，通信にかかる労力を減らすことも誤り訂正符号の目的である．労力にあたる性能の指標は，通信路を使う回数に対してどれだけ情報を送れるかを示す符号化率 R がある [5]．R は，

$$R \triangleq \frac{\log_2 |C|}{\log_2 |\mathcal{X}^n|} \tag{2}$$

で定義される．ここで，C は符号であり，$|C|$ は符号語の数である．$\mathcal{X}$ は通信路アルファベットであり，2 元対称通信路の場合，$\mathcal{X} = \{0, 1\}$ である．$\mathcal{X}^n$ は有限集合 $\mathcal{X}$ からなる長さ n の系列の全ての集合である．n は正の整数である．有限集合 A に含まれる要素数を $|A|$ と表記する．また，復号の際にどれだけ計算量を減らせるかということも考える必要がある．シャノンによって誤り訂正符号の理論的な性能の限界値は示されている．しかし，シャノンは限界を示しただけで，実際に誤り訂正符号の作り方を示したわけではないため，シャノンの限界値を達成するためにリード・ソロモン符号，畳み込み符号，LDPC 符号など様々な誤り訂正符号が開発された [5].

誤り訂正符号の性能は，送る情報をどうやって工夫して送るかを考える符号化と，受け取った受信語からどうやって送られてきた情報を復元するかを考える復号によって決まる．繰り返し符号では，送る情報を 3 回繰り返すことが符号化にあたり，受信語を見て多いほうの数字が送られてきたと判定することが復号化にあたる．符号化，復号化の方法は一通りではなく，どうやって行えばより良い性能を得られるかを考えることが必要になる．ただし，3 章で詳しく述べるが本研究では，符号化の方法としてラテン方陣のルールを用いることを前提としている．そのため，符号化の方法が最適であるかどうかは議論せず，復号手法によって誤り訂正符号の性能を上げることについて議論する．

2.2 最尤復号法の原理

符号語を $\boldsymbol{x}$，受信語を $\boldsymbol{y}$ とする．1.1 節で述べたように，通信を確率的問題と考える．通信路への入力信号を表す確率変数を X，通信路から復号器への出力信号を表す確率変数を Y とする．受け取った受信語をみて，送られてきた符号語がなんであったかを推測するという事は，$\boldsymbol{y}$ が与えられたもとでの $\boldsymbol{x}$ の確率，すなわち条件付確率 $P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})$ を計算し，送られてきた符号語は $P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})$ の値が最も大きくなる $\boldsymbol{x}$ であると判定することである．式で表すと，復号器では

$$\arg\max_{\boldsymbol{x}\in C} P_{X|Y}(\boldsymbol{x}|\boldsymbol{y}) \tag{3}$$

を計算する．これをブロック MAP 復号法という．C は符号語の集合であり，$\boldsymbol{x} \in C$ である．ベイズ則を使って式 (3) を変形すると

$$P_{X|Y}(\boldsymbol{x}|\boldsymbol{y}) = \frac{P_X(\boldsymbol{x})P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})}{P_Y(\boldsymbol{y})} \tag{4}$$

という式が得られる．式 (4) で，$P_X(\boldsymbol{x})$ は符号語の確率分布，$P_Y(\boldsymbol{y})$ は受信語の確率分布である．$P_Y(\boldsymbol{y})$ は受信語 $\boldsymbol{y}$ にのみ依存する値なので $P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})$ を比較する際に全て同じ値になるので，式 (3) の計算に必要のない値となる．また，M を符号語の総数とし，$\boldsymbol{x}$ の生起確率を

$$P_X(\boldsymbol{x}) = \frac{1}{M}\mathbb{1}[\boldsymbol{x} \in C] \tag{5}$$

とする．$\mathbb{1}[condition]$ は $condition$ が真なら 1，偽なら 0 となる関数である．したがって，式 (5) は全ての符号語の生起確率は一定であり，符号語以外の生起確率は 0 であることを意味する．式 (4) を求める際に $P_X(\boldsymbol{x})$ の値は $\boldsymbol{x}$ に依らない定数となる．したがって，復号器は

$$\arg\max_{\boldsymbol{x}\in C} P_{Y|X}(\boldsymbol{y}|\boldsymbol{x}) \tag{6}$$

を求めることで復号できるようになる．条件付確率 $P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})$ は，$\boldsymbol{x}$ を送った際に，$\boldsymbol{y}$ が届く確率，すなわち通信路そのものである．従って，符号語の生起確率 $P_X(\boldsymbol{x})$ が等確率であり，

復号器が通信路の性質をわかっている場合に (6) を計算することで復号できる．この復号方法を最尤復号法という．符号語の生起確率が等確率であれば，最尤復号法は全ての復号法の中で最小の復号誤り確率となる．そのため，最も復号の成功確率を上げるという点では最適な方法となる．しかし，式 (6) からわかるように，最尤復号法はまず全ての符号語を用意し，そのすべての $\boldsymbol{x}$ について $P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})$ を計算してから比較するという手順を踏む必要があるため，その計算量が現実的に実現可能なのかということ，全ての符号語を用意しておく領域が必要となるといった事が問題となる．

2.3 sum-product アルゴリズムの原理

最尤復号法に対し，計算を実用的なレベルで高速に行うことができるアルゴリズムとして sum-product アルゴリズムがある [5]．本節では文献 [5] に基づいて，sum-product アルゴリズムの概要について述べる．2.1 節においてブロック MAP 復号では式 (3) を計算することによって復号を行った．ここで，送信する符号語 $\boldsymbol{x}$ が符号長 n の符号であるとする．送信符号語を表す確率変数を $X = (X_1, X_2, \ldots, X_n)$ とすると $P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})$ は，

$$P_{X|Y}(\boldsymbol{x}|\boldsymbol{y}) = P_{X_1,X_2,\ldots,X_n|Y}(x_1, x_2, \ldots, x_n|\boldsymbol{y}) \tag{7}$$

となる．変数集合 V を $V \triangleq \{x_1, x_2, \ldots, x_n\}$ と定義する．ある送信シンボル $X_i (i \in [1, n])$ に着目する．$[1, n]$ は 1 から n までの連続した整数の集合である．受信語 $\boldsymbol{y}$ を受け取ったもとでの符号語のシンボル x_i の確率は，

$$P_{X_i|Y}(x_i|\boldsymbol{y}) = \sum_{V \setminus x_i} P_{X_1,X_2,\ldots,X_n|Y}(x_1, x_2, \ldots, x_n|\boldsymbol{y}) \tag{8}$$

となり，式 (7) を周辺化することでシンボルごとの条件付確率が計算できる．シンボルごとの条件付確率を求め，シンボルごとに復号を行う方法を，シンボル MAP 復号法という．シンボル MAP 復号法はシンボルの誤り確率を最小にする復号方法である．$P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})$ は式 (4) で書ける．いま，通信路が定常無記憶通信路であるとし，1 シンボルずつ通信路を通して送信を行うとすれば，$P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})$ は，

$$P_{Y|X}(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{n} P_{Y_i|X_i}(y_i|x_i) \tag{9}$$

となる．また，符号語 $\boldsymbol{x}$ の生起確率が等確率，すなわち式 (5) で表されるとする．ここで，符号語であるかどうかをチェックする $\mathbb{1}[condition]$ をシンボルごとにチェックする関数に書き換える事を考える．例として，符号長が 4 の場合について考える．符号語が $\mathbb{1}[x_1$ と x_2 がルール 1 を満たす$]$, $\mathbb{1}[x_1$ と x_3 がルール 2 を満たす$]$, $\mathbb{1}[x_3$ と x_4 がルール 3 を満たす$]$ と 3 つのルールでチェックされていると考える．言い換えると，x_1 と x_2, x_1 と x_3, x_3 と x_4 をそれぞれ

チェックするルールを考え，全てのルールを満たしていれば符号語とする，というような符号を作るということである．この時，それぞれのルールを

$$f_1(x_1, x_2) = \mathbb{1}[x_1 \text{と } x_2 \text{がルール 1 を満たす}] \tag{10}$$

$$f_2(x_1, x_3) = \mathbb{1}[x_1 \text{と } x_3 \text{がルール 2 を満たす}] \tag{11}$$

$$f_3(x_3, x_4) = \mathbb{1}[x_3 \text{と } x_4 \text{がルール 3 を満たす}] \tag{12}$$

と定義する．この時，$P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})$ は，式 (4), (5), (9) より

$$\begin{aligned} P_{X|Y}(\boldsymbol{x}|\boldsymbol{y}) &= \frac{P_X(\boldsymbol{x})P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})}{P_Y(\boldsymbol{y})} \\ &= \frac{1}{M}\frac{f_1(x_1,x_2)f_2(x_1,x_3)f_3(x_3,x_4)\prod_{i=1}^4 P_{Y_i|X_i}(y_i|x_i)}{P_Y(\boldsymbol{y})} \\ &= K f_1(x_1,x_2)f_2(x_1,x_3)f_3(x_3,x_4)\prod_{i=1}^4 P_{Y_i|X_i}(y_i|x_i) \end{aligned} \tag{13}$$

と書くことができる．ここで，$K = \frac{1}{MP_Y(\boldsymbol{y})}$ である．符号語のシンボル x_1 に着目し，$P_{X_1|Y}(x_1|\boldsymbol{y})$ を求めると，式 (8), (13) より，

$$\begin{aligned} P_{X_1|Y}(x_1|\boldsymbol{y}) &= \sum_{V\setminus x_1} K f_1(x_1,x_2)f_2(x_1,x_3)f_3(x_3,x_4)\prod_{i=1}^4 P_{X_i|Y_i}(x_i|y_i) \\ &= K\sum_{V\setminus x_1} f_1(x_1,x_2)f_2(x_1,x_3)f_3(x_3,x_4)\prod_{i=1}^4 P_{X_i|Y_i}(x_i|y_i) \end{aligned} \tag{14}$$

となる．ここで，$V = \{x_1, x_2, x_3, x_4\}$ である．K は x_1 に依らない定数であるから，$P_{X_1|Y}(x_1|\boldsymbol{y})$ を比較する際に必要ない．式 (14) をこのまま計算する場合，全ての符号語について和をとる必要があるため，最尤復号法と同じように符号語数に比例した計算量が必要になる．

周辺化の計算について，分配則の考え方を用いることで，計算量を減らすことができる．いま，関数 $g(x_1, x_2, x_3, x_4)$ を考える．関数が

$$g(x_1, x_2, x_3, x_4) = g_1(x_1)g_2(x_1, x_2)g_3(x_1, x_3)g_4(x_3, x_4) \tag{15}$$

と因子分解できるとする．ここで，$g(x_1, x_2, x_3, x_4)$ の x_1 についての周辺化関数を

$$h_1(x_1) \triangleq \sum_{V\setminus x_1} g(x_1, x_2, x_3, x_4) \tag{16}$$

と定義する．ここで，$V = \{x_1, x_2, x_3, x_4\}$ である．この時，

$$h_1(x_1) = \sum_{x_2}\sum_{x_3}\sum_{x_4} g_1(x_1)g_2(x_1, x_2)g_3(x_1, x_3)g_4(x_3, x_4) \tag{17}$$

である．式 (15) をこのまま計算するのではなく，分配則を用いて

$$h_1(x_1) = g_1(x_1)\left(\sum_{x_2} g_2(x_1,x_2)\right)\left(\sum_{x_3} g_3(x_1,x_3)\left(\sum_{x_4} g_4(x_3,x_4)\right)\right) \tag{18}$$

と変形して計算することで計算量を少なくすることができる．この考え方を利用して，式 (14) の定数 K を除いた部分を変形すると，

$$\begin{aligned}
&\sum_{V\setminus x_1} f_1(x_1,x_2)f_2(x_1,x_3)f_3(x_3,x_4)\prod_{i=1}^{4} P_{Y_i|X_i}(y_i|x_i) \\
=&P_{Y_1|X_1}(y_1|x_1)\left(\sum_{x_2} f_1(x_1,x_2)P_{Y_2|X_2}(y_2|x_2)\right) \\
&\left(\sum_{x_3} f_2(x_1,x_3)P_{Y_3|X_3}(y_3|x_3)\left(\sum_{x_4} f_3(x_3,x_4)P_{Y_4|X_4}(y_4|x_4)\right)\right)
\end{aligned} \tag{19}$$

とできる．このようにシンボル MAP 復号で分配則を利用することで，最尤復号法よりも計算量を少なくできる．

式 (18) のような，分配則の計算の様子を，図 4 に示すような tree 構造としてモデル化して表すことができる．tree 構造の見方を説明する．○は変数 x_i を表しており，変数ノードという．■は関数 g_i を表しており，因子ノードという．まず，周辺化する際に着目する変数ノードを根とする．式 (18) の場合 x_1 について周辺化するため x_1 の○を根とする．次に，根となる変数ノードに関する因子ノードである■を根となる○の下に書き線でつなぐ．○と■をつなぐ線を枝と呼ぶ．式 (18) の場合，x_1 に関する g_i は g_1, g_2, g_3 であるから，その 3 つにたいして前述した処理を行う．次に，根につないだ■について，上につながっている○以外でチェックする変数があればその下に○を描き枝でつなぐ．○を描き終わった後，同様にその○について上につながっている■以外にその変数をチェックする関数があればその下に■を描き枝でつなぐ．この処理を繰り返し続け，全てのノードに関して新たに○と■が下に現れなくなったら

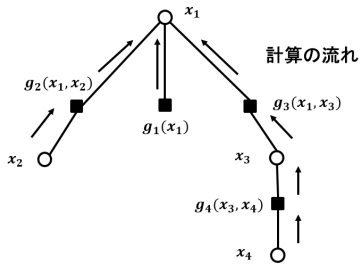


図 4　分配則のモデル化

処理を終える．ある変数に着目して，その変数から見て上につながっているノードを親ノードといい，下につながっているノードを子ノードという．tree の下に何も枝がつながっていないノードを葉ノードという．式 (18) の計算を行う場合，まず $\sum_{x_4}$ の計算を行い，終わったら $g_2(x_1, x_3)$ に乗算し $\sum_{x_3}$ の計算を行うというように内側の () から計算していくが，その様子を図 4 では一番下のノードから上のノードに向かって上がっていくと表している．○から■に向かう枝が $\sum$ の計算を表しており，■から○に向かう枝が $\prod$ の計算を表している．図 4 のように計算を図示すると，(18) の計算は下のノードから根となるノードへメッセージを送っているというイメージになる．

ここまで式 (18) で表される場合の関数の例について述べたが，これらの処理の流れを一般化する．変数ノードから因子ノードへの処理を変数ノード処理とする．変数ノード処理では変数ノード x_k から因子ノード g_i へのメッセージ (局所積) を

$$M_{x_k \to g_i}(x_k) = \prod_{a \in N(x_k) \setminus g_i} M_{a \to x_k}(x_k) \tag{20}$$

とする．ここで，ノードを v とし，v に隣接するノードの集合を $N(v)$ としている．ただし，x_k が葉ノードの場合には，初期条件として，$M_{x_k \to g_i}(x_k) = 1$ とする．$M_{x_k \to g_i}(x_k)$ は $x_k \to g_i$ 方向に伝達されるメッセージを表しており，(x_k) の関数となっている．例として，図 5 に示すような変数ノードに 3 つの因子ノードがつながっている場合を考える．自分の子ノードである g_j と g_l から送られてきたメッセージである,$M_{g_j \to x_k}$, $M_{g_l \to x_k}$ の積を取り, 親ノードである g_i にメッセージを送るという処理になっている. 次に，因子ノードから変数ノードへの処理を因子ノード処理とする．因子ノード処理では，因子ノード g_i から変数ノード x_k へのメッセージ (局所積和) を

$$M_{g_i \to x_k}(x_k) = \sum_{N(g_i) \setminus x_k} g_i(Ai) \prod_{a \in N(g_i) \setminus x_k} M_{a \to x_k}(a) \tag{21}$$

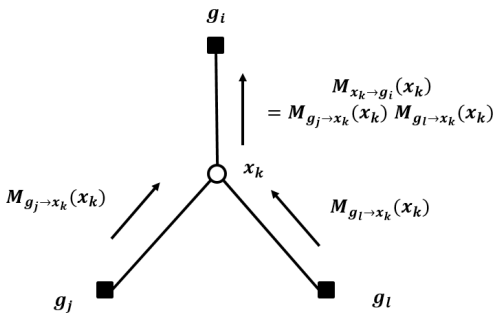


図 5　変数ノード処理

とする．ただし，g_i が葉ノードの場合，初期条件として $M_{g_i \to x_k}(x_k) = g_i(x_k)$ とする．ここで，変数の集合を $V \triangleq \{x_1, x_2, \ldots, x_n\}$ とし，A_i は g_i がチェックする変数の集合であり，V の部分集合である．変数ノード処理と同様に，因子ノードから変数ノードに伝達されるメッセージを表した式となっている．例として，図 6 に示すような変数ノードに 3 つの因子ノードがつながっている場合を考える．因子ノードでは，子ノードである x_j, x_l から送られてきたメッセージ $M_{x_j \to g_i}$, $M_{x_l \to g_i}$, と関数 $g_i(x_j, x_k, x_l)$ の積を取り，親ノードである，x_k 以外の変数で周辺化を行い，親ノードにメッセージを送るという処理になっている．

変数ノード処理と因子ノード処理を tree の下から上に向かって行い，全ての子ノードから根ノードにメッセージが到達した時点で次の周辺化関数の計算を行う．根ノード x_r において，

$$M_{x_r}(x_r) = \prod_{a \in N(x_r)} M_{a \to x_r}(x_r) \tag{22}$$

を計算する．根ノードでは自分の子ノードから到達したすべてのメッセージの積を計算する．$M_{x_r}(x_r)$ は求めたい周辺化関数 $h_r(x_r)$ と一致する．このようにして任意の符号について，シンボルごとの条件付確率 $P_{X_1|Y}(x_1|\boldsymbol{y})$ が式 (17) から式 (18) に変形したように分配則が適用可能な式の形になっていれば，シンボル MAP 復号を行う際の計算量を減らすことができる．このようにして計算を行うアルゴリズムが sum-product アルゴリズムである．

例として式 (15) の x_1 の周辺化についての tree を図 4 に示したが，x_2 についても同様に周辺化を行うための tree を描いた場合，図 7 のような tree が描ける．図 7 の tree を見ると，計算の一部が図 4 の x_1 についての周辺化の計算と同じであることがわかる．そのような重複する計算を省き，さらに計算量を減らすアルゴリズムが同時更新 sum-product アルゴリズムである．

同時更新 sum-product アルゴリズムの適切な手順は符号の構造により異なるが，以下では本研究における手順を述べる．同時更新 sum-product アルゴリズムの流れは図 8 に示すファクターグラフによって表される．式 (15) の関数について同時更新 sum-product アルゴリズム

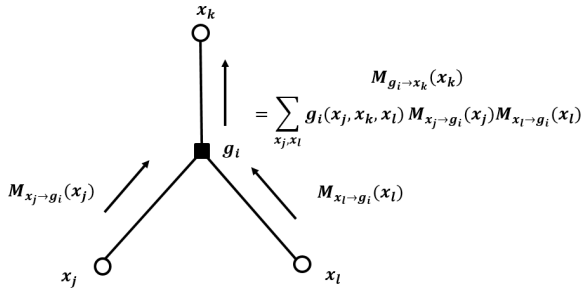


図 6　因子ノード処理

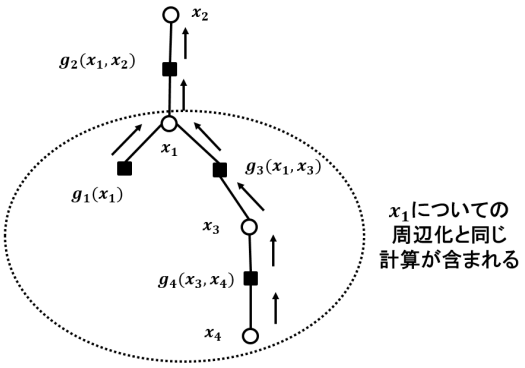


図 7　式 (15) の x_2 の周辺化確率を求める場合の tree 構造

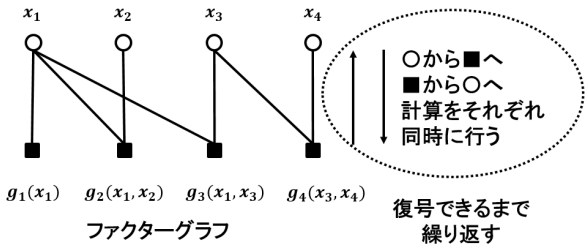


図 8　ファクターグラフ

を行う．図 8 のように変数ノード○を上側，因子ノード■を下側に全て列挙して並べ，それぞれを枝でつなぐ．同時更新 sum-product アルゴリズムのフローチャートを図 9 に示す．まず，ファクターグラフの全ての枝について，○から■への変数ノード処理を行う．次に，ファクターグラフの全ての枝について，■から○への因子ノード処理を行う．最後に全ての変数について式 (22) の周辺化計算を行う．計算値を見てシンボルごとの復号を行い，符号語になっていればそれを出力し終了，なっていなければ変数ノード処理の手順まで戻り同じ処理を繰り返し行う．これを符号語が出力されるまで繰り返す．このように計算することで，重複する枝の計算を省略することでき，計算量を減らすことができる．

sum-product アルゴリズムは計算を高速化でき，正確な周辺化関数を得ることができるので，様々な分野で用いられる．しかし，ファクターグラフ内にループが含まれる場合，正確な計算値を得ることができないことがわかっている．ファクターグラフ内のループと，ループが計算に及ぼす影響について述べる．

まず，ファクターグラフ内にループが無い場合について考える．図 10 のような tree 構造を持つ符号について図 9 の sum-product アルゴリズムを適用する．図 10 の符号について x_1 の

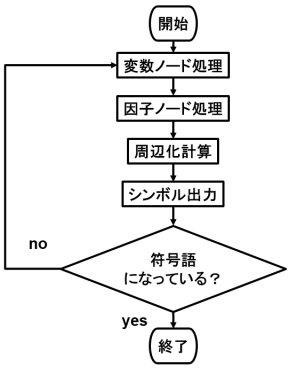


図 9　sum-product アルゴリズムのフローチャート

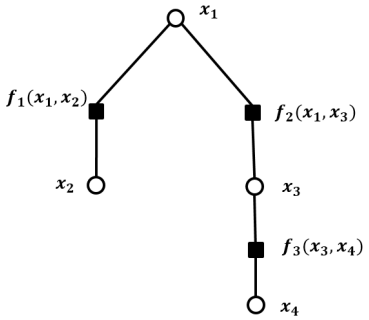


図 10　ファクターグラフにループが無い符号の例

周辺化関数を求める．図 9 に示すように，sum-product アルゴリズムは符号語が出力されるまで計算を繰り返す．計算の繰り返しごとに出力される x_1 の周辺化計算値は図 11 のようになる．1 回目の計算では変数ノード処理，因子ノード処理を 1 回行うため x_1 から見ると x_2 と x_3 からのメッセージを受け取ることができる．そのように，tree 構造の深さは sum-product アルゴリズムの繰り返しの回数に対応していることがわかる．1 回目の計算による周辺化計算値は図 10 の tree 構造と一致しておらず，1 回目の計算では正しい計算値を得られていないことがわかる．1 回目の周辺化計算値で符号語にならなかった場合 2 回目の計算を行う．その場合，変数ノード処理と因子ノード処理の回数が増えるため，図 11 に示すように x_4 からのメッセージが x_3 のノードに渡され，そのメッセージを x_1 は受け取ることができる．左側の x_2 の側の枝では，x_2 から見て子ノードが存在しないため，受け取るメッセージが無い．そのため，計算を繰り返してもこれ以上下に枝が伸びていかない．2 回目の計算による周辺化計算値は図 10 の tree 構造と一致しており，x_1 の周辺化計算値に関しては，2 回目の計算で正しい計算値を得られることがわかる．2 回目の周辺化計算で符号語にならなかった場合も 3 回目の計算を

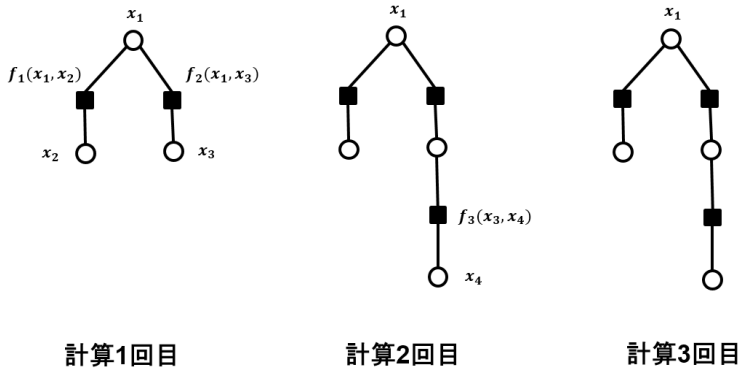


図 11　sum-product アルゴリズムによって図 10 の符号で出力される x_1 の周辺化計算値

行う．左側の枝は 2 回目の計算ですでにメッセージの受け渡しが終了しているため，3 回目の計算でも同様に枝は伸びることは無い．右側の x_3 側の枝は x_4 以降の子ノードが存在しないため，計算 3 回目では新たに枝が伸びない．したがって，3 回目の計算で得られる周辺化計算値は 2 回目の計算値と同じであり，正しい周辺化計算値となっていることがわかる．4 回目以降の計算も同様である．このように，sum-product アルゴリズムによって図 10 の符号の x_1 の周辺化計算値を求める場合，sum-product アルゴリズムによる計算を繰り返していくと正しい計算値が得られ，計算値が収束していくことがわかる．

次に，ファクターグラフ内にループを持つ符号の例について考える．図 12 に tree 構造にループが存在する符号の例を示す．図 12 は x_1 の周辺化計算を行うための tree となっている．x_1 の子ノードとなる因子ノードは f_1 と f_2 の二つである．f_1 と f_2 の子ノードは x_2 となっている．図 12 に示すように，x_1 から下に向かって枝をたどっていった場合に，再び x_1 に戻ってきてしまう構造をループという．図 12 の符号について x_1 の周辺化計算を行うために sum-product アルゴリズムを適用する．計算の繰り返しごとに出力される x_1 の周辺化計算値は図 13 のようになる．1 回目の計算を行うと，まず左側の枝について f_1 は子ノードである x_2 から送られてきたメッセージを受け取り，x_1 にメッセージを送ろうとする．しかし，右側

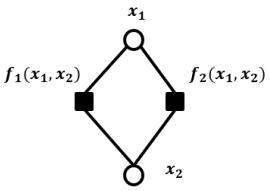


図 12　ファクターグラフにループがある符号の例

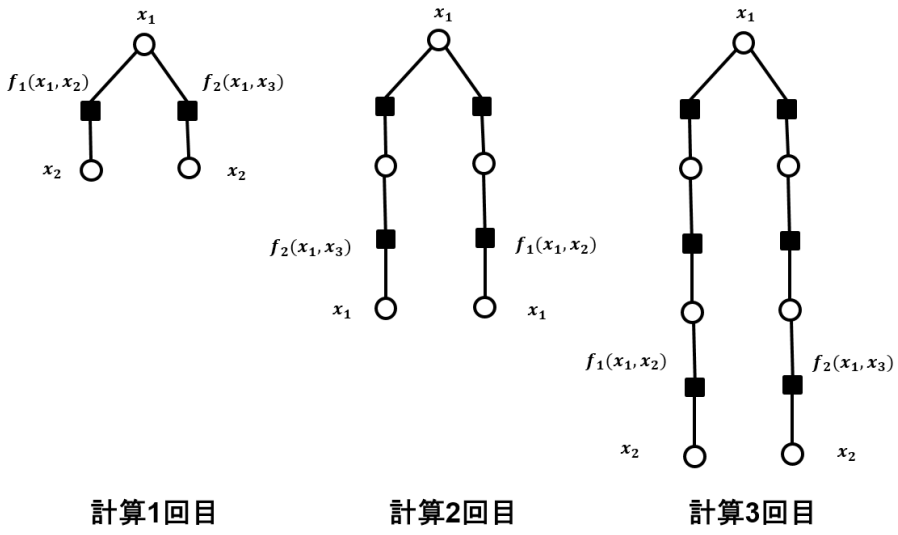


図 13　sum-product アルゴリズムによって図 12 の符号で出力される x_1 の周辺化計算値

の枝でも f_2 の子ノードが x_2 であるから，x_2 からのメッセージを受け取って x_1 にメッセージを送ろうとする．したがって，本来であれば図 12 の tree 構造の周辺化計算値を求めたかったのに，sum-product アルゴリズムを行った場合に出力されるのは図 13 のような枝分かれした tree 構造の計算値となってしまう．1 回目の計算で符号語が出力されなかった場合 2 回目の計算を行うが，tree にループが無い場合図 11 のように計算値が収束する．しかし，ループが存在する場合，sum-product では 1 回目の計算の tree の一番下のノードから見た子ノードを探そうとするので，左側の枝については f_1 を親ノードとしたときの x_2 から見た子ノードは f_1 であり，さらにその子ノードは x_1 であるから x_1 からのメッセージを受け取ろうとする．右側の枝も同様に，f_1 を通して x_1 からのメッセージを受け取ろうとする．したがって，図 13 のような周辺化計算値が出力されてしまう．3 回目以降の計算も同様の傾向で周辺化計算が行われる．つまり，ファクターグラフにループが存在する符号に sum-product アルゴリズムを適用した場合，アルゴリズムは符号のループに対して無限に下に続く tree 構造になっていると勘違いをしたふるまいをする．その結果，正しい周辺化関数を得ることができず，さらに計算値が収束しないという不具合が起こる．これがループの影響である．逆に言うと，ループを持つ符号の場合，求めたい周辺化計算値について式 (18) のように $\sum$ の計算を分配することができない．そのため，tree 構造にループが存在する場合，sum-product アルゴリズムは，求めたい符号の tree 構造ではなく，sum-product アルゴリズムが適用できるように別の tree 構造を無理やり作って，別の計算値を求めていると言える．

3　ラテン方陣の誤り訂正

2 章では一般的な誤り訂正符号の復号方法について述べたが，本章では実際にラテン方陣を誤り訂正符号と考えて議論する．

3.1　問題設定

初めに，ラテン方陣のルールを説明する．ラテン方陣の概要を図 14 に示す [6]．まず，$n \times n$ マスの正方形の表を用意する．n の数は任意だが，本研究では 4 の場合について考える．次に，行と列で重複する文字が無いように表に文字を入れたものがラテン方陣である．本研究では，図 14 に示すように，受信者は送られてきた 4 行 4 列の表を見て，ルールを満たすように正しく表を訂正するという問題設定とする．

まず，ラテン方陣を誤り訂正符号として用いることができる理由について述べる．図 15 に 2.1 節で述べた繰り返し符号とラテン方陣の比較を示す．繰り返し符号では，送りたい情報を 3 回繰り返して符号語とする．ラテン方陣を誤り訂正符号として用いる場合，ラテン方陣が符号語に対応する．つまり，あるラテン方陣に対して，このラテン方陣は情報 1 を示している，というように割り当てることである．次に，符号語を通信路を通して 1 ビットずつ受信者に送る．通信では雑音の影響を受けてしまい，受信者は送られてきた符号語とは異なる数字列を受け取ってしまう場合がある．ラテン方陣の場合も同様に，1 マスずつ通信路を通して相手に送り，通信路を通った際に雑音の影響で，表内の数字が変わってしまうと考える．通信路で誤りが発生した場合，受信者は送られてきた符号語と異なる受信語を受け取ってしまうが，受け取った受信語が符号語でなければ，通信路で誤りが生じたと判断することができる．また，通信路で誤りが生じ数字が反転してしまう確率が低ければ，000 が 2 か所反転して 101 になってしまう確率よりも，111 が 1 か所反転して 101 になった確率のほうが高いと考えることができる．そのため，受信語を見て送られてきた符号語を推測することができる．ラテン方陣の場合

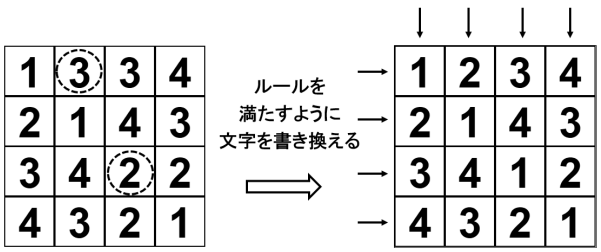


図 14　ラテン方陣

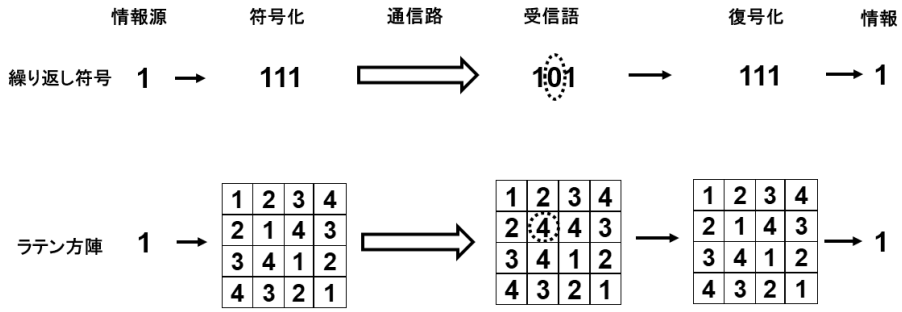


図 15　繰り返し符号とラテン方陣の比較

も同様であり，受信語がラテン方陣のルールを満たしていなければ，通信路で誤りが生じたと判断できる．また，受信語をラテン方陣のルールを満たすように書き換えることで，ラテン方陣を復元することができ，情報を復元することができる．このように考えると，ラテン方陣は繰り返し符号と同じ誤り訂正符号であると考えることができる．

次に，ラテン方陣の復号の考え方の概要を述べる．前述したように，通信路で誤りが発生する確率が低ければ，受信語の数字が反転してしまった確率も低いと考えることができる．したがって，受信語と異なる箇所が少ない，できるだけ少ない訂正回数でラテン方陣のルールを満たすように復号することが，誤り確率を低くする復号方法となる．図 16 に 1 マス誤ってしまった表を受信した場合の復号を示す．まず，1 マス誤った表の場合，2 行目と 2 列目がルールを満たしていないため，ラテン方陣になっていないことがわかり，符号語ではないため通信

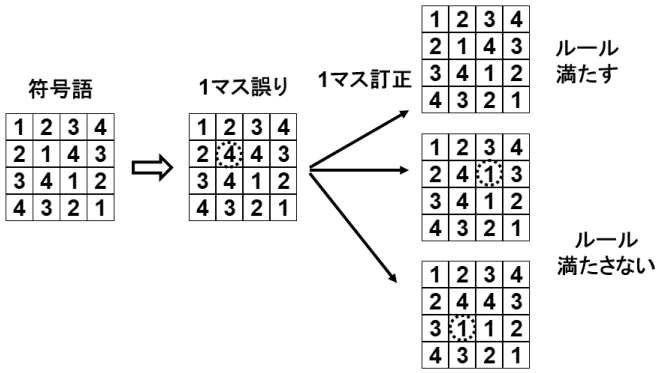


図 16　1 マス誤ってしまった場合の復号の例

路で誤りが発生していることがわかる．まず，1 マス訂正でラテン方陣に戻すことができないかを考える．元のラテン方陣から 1 マス誤っているのが受信語なので，当然 1 マス訂正で元のラテン方陣に戻すことができる．少なくとも 1 回訂正で元に戻るものが一つ存在するため，他に 1 マス訂正でラテン方陣に戻るものが無いかを探す．元のラテン方陣に戻すために 2 行 2 列目のマスを訂正したが，2 行目や 2 列目の他のマスを訂正することでラテン方陣にならないかを考える．2 行目の 3 列目を 1 に訂正した場合，2 行目はルールを正しく満たしているが，2 列目や 3 列目がルールを満たさない．2 列目の 3 行目を訂正した場合も同様であり，2 列目のルールを満たすことはできるが，2 行目と 3 行目がルールを満たさない．また，2 行目や 2 列目以外のマスを訂正すると，当然ルールを満たしていなかった 2 行目や 2 列目はルールを満たしていないままであるためラテン方陣に戻らない．このように考えていくと，1 回訂正でラテン方陣に戻るものは元のラテン方陣のみあることがわかり，元のラテン方陣に復元することができる．図 16 の例だけでなく，他のラテン方陣を符号語とした場合や，他のマスが 1 か所異なる数字に変わってしまっている場合も同様の考え方で復号することができる．

次に復号ができない場合について考える．図 17 に 2 マス誤ってしまった表を受信した場合の例について示す．受信語はラテン方陣のルールを満たしていないため，まず 1 回訂正でラテン方陣になるものがないか考える．図 17 に示すように，2 行目では 4 が重複しており，3 行目では 1 が重複しているため，ルールを満たしていない．1 回訂正すれば，片方の行のルールを満たすことができるが，訂正しない側の行のルールを満たすことができないため 1 回の訂正では復号することができない．最小の訂正回数が 1 回ではないため，次は 2 回訂正でラテン方陣になるものがないか考える．2 回訂正でラテン方陣になるものを考えた場合，図 17 に示すように，2 回訂正でルールを満たすものが少なくとも 2 つ存在する．したがって，最小の訂正回

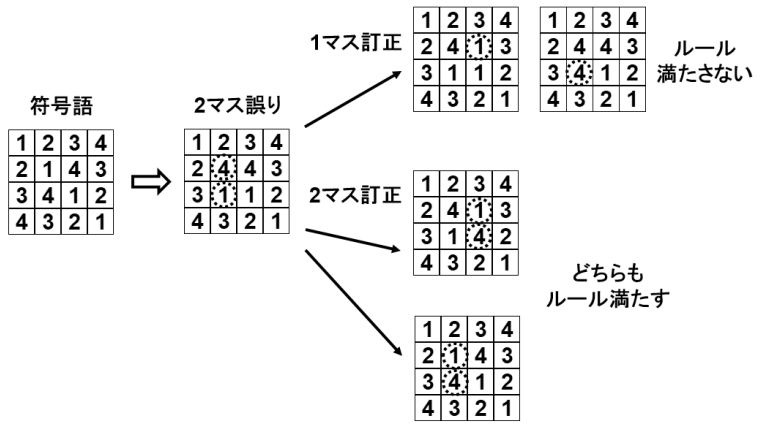


図 17　2 マス誤ってしまった場合の復号の例

数のものが複数存在するため，元のラテン方陣がどちらであったかを区別することができず，復号することができない．図 17 はあくまで復号できない例であり，誤った個所によっては 2 か所誤りであっても復号可能なものが存在する．本研究では，ラテン方陣の復号の考え方の例として述べるにとどめ，誤りの個数と訂正能力の関係についてはこれ以上議論しない．

ここまでラテン方陣を誤り訂正符号とする場合の考え方について述べたが，ここからは実際に数学的な問題設定を定義する．まず，送信者は送りたい情報を符号化する．その符号語としてラテン方陣を用いるということは図 18 のように送りたい情報源に符号語を割り当てるということになる．

4 行 4 列のラテン方陣は 576 個存在する [6]．従って，1 個のラテン方陣で

$$\log_2 576 \fallingdotseq 9.17[bit] \tag{23}$$

の情報を送ることができる．符号語としてのラテン方陣を一般化すると図 19 のように書くことができる．送信する符号語は $\boldsymbol{x} = x_1, x_2, \ldots, x_{16}$ であり，16 個のシンボルからなる．符号語のシンボル $x_i (i \in [1, 16])$ の取りうる値は $1, 2, 3, 4$ である．$f_i (i \in [1, 8])$ は各シンボルがルールを満たしているかどうかをチェックする関数であり 8 つある．sum-product 復号では因子ノードにあたる．f_n はそれぞれ，

$$f_1 = f_1(x_1, x_2, x_3, x_4) \triangleq \mathbb{1}[x_1, x_2, x_3, x_4\text{がルールを満たす}] \tag{24}$$

$$f_2 = f_2(x_5, x_6, x_7, x_8) \triangleq \mathbb{1}[x_5, x_6, x_7, x_8\text{がルールを満たす}] \tag{25}$$

$$f_3 = f_3(x_9, x_{10}, x_{11}, x_{12}) \triangleq \mathbb{1}[x_9, x_{10}, x_{11}, x_{12}\text{がルールを満たす}] \tag{26}$$

$$f_4 = f_4(x_{13}, x_{14}, x_{15}, x_{16}) \triangleq \mathbb{1}[x_{13}, x_{14}, x_{15}, x_{16}\text{がルールを満たす}] \tag{27}$$

$$f_5 = f_5(x_1, x_5, x_9, x_{13}) \triangleq \mathbb{1}[x_1, x_5, x_9, x_{13}\text{がルールを満たす}] \tag{28}$$

$$f_6 = f_6(x_2, x_6, x_{10}, x_{14}) \triangleq \mathbb{1}[x_2, x_6, x_{10}, x_{14}\text{がルールを満たす}] \tag{29}$$

$$f_7 = f_7(x_3, x_7, x_{11}, x_{15}) \triangleq \mathbb{1}[x_3, x_7, x_{11}, x_{15}\text{がルールを満たす}] \tag{30}$$

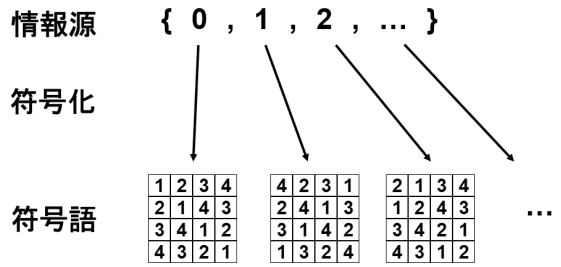


図 18　ラテン方陣による符号化

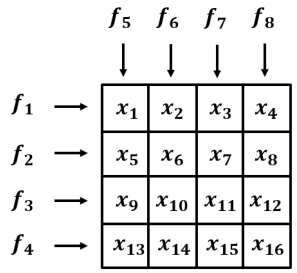


図 19　符号語としてのラテン方陣

$$f_8 = f_8(x_4, x_8, x_{12}, x_{16}) \triangleq \mathbb{1}[x_4, x_8, x_{12}, x_{16}\text{がルールを満たす}] \tag{31}$$

と書ける．また，$V = \{x_1, x_2, \ldots, x_{16}\}$ とする．

符号語の集合を C とする．符号語の総数を $|C|$ とすると，4 行 4 列のラテン方陣の場合 $|C| = 576$ であるから符号語の生起確率は，

$$\begin{aligned} P_X(\boldsymbol{x}) &= \frac{1}{|C|}\mathbb{1}[\boldsymbol{x} \in C] \\ &= \frac{1}{576}\prod_{i=1}^{8} f_i \end{aligned} \tag{32}$$

で，等確率とする．符号語の各シンボルは図 20 の通信路を使って送信する．受信語を

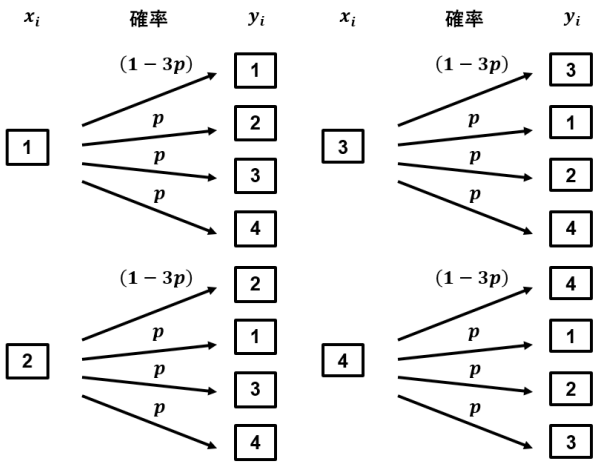


図 20　ラテン方陣の通信路

$\boldsymbol{y} = y_1, y_2, \ldots, y_{16}$ とする．すると，

$$P_{Y_i|X_i}(y_i|x_i) = \begin{cases} 1-3p & (y_i = x_i) \\ p & (y_i \neq x_i) \end{cases} \tag{33}$$

となる．上記の通信路を通して, シンボルごとに受信者に送る．通信路は定常無記憶であるとする．よって，

$$P_{Y|X}(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{n} P_{Y_i|X_i}(y_i|x_i) \tag{34}$$

が成り立つ．

3.2 最尤復号法

3.1 節で定義した符号としてのラテン方陣について，実際に最尤復号法を適用する場合について考える．いま，符号語の生起確率 $P_X(\boldsymbol{x})$ は等確率であるので，式 (6) によって復号する．$P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})$ は，式 (34) で表されるので，復号器は推定符号語 $\hat{\boldsymbol{x}}$ を，

$$\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x} \in C} \prod_{i=1}^{n} P_{Y_i|X_i}(y_i|x_i) \tag{35}$$

として出力する．2.1 節で述べたように，最尤復号法による復号は復号誤り確率を最小とするため，誤り確率的には最も良い復号方法となる．しかし，符号語数に比例した計算回数が必要となる．4 行 4 列のラテン方陣は 576 個であるが，5 行 5 列のラテン方陣は 161280 個に増加する [7]．本研究では簡単のため 4 行 4 列の場合について考えているが，n 行 n 列に一般化する事を考えた場合，行と列の数に応じて計算量が膨大に増えると考えられる．また，全ての符号語 C について，式 (35) を計算する必要があるため，事前に全てのラテン方陣を用意しておく領域が必要となる．

3.3 sum-product 復号

sum-product アルゴリズムを適用する場合について考える．例として x_1 についての周辺化関数を求める．前提条件は 3.1 節で前述したとおりであり，式 (14) の例に従い定数 K を簡略化した計算値を求める．そうすると，ラテン方陣における求めたい周辺化計算値は，式 (32), (34) を用いて，式 (13), (14) の例と同様に考えると

$$\sum_{V \setminus x_1} \prod_{j=1}^{8} f_j \prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i) \tag{36}$$

となる．式 (36) に従い x_1 を求める場合の tree 構造を描くと図 21 となる．ラテン方陣はファ

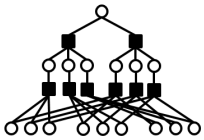


図 21　ラテン方陣の x_1 の tree 構造

クターグラフ内にループが存在する符号となっている．従って，式 (36) は，式 (17) から式 (18) に変形したように分配則を用いて変形することができない．ファクターグラフ内のループの例を図 22 に示す．

ラテン方陣は，図 22 のようなループがファクターグラフ内に複数存在する符号語となっている．ラテン方陣に図 9 の sum-product アルゴリズムを適用した場合の，計算の繰り返しごとに出力される x_1 の周辺化計算値は図 23 に示す tree 構造の計算値となる．1 回目の計算により求まる計算値は

$$\begin{aligned} & P_{Y_1|X_1}(y_1|x_1) \sum_{x_2,x_3,x_4} f_1 P_{Y_2|X_2}(y_2|x_2) P_{Y_3|X_3}(y_3|x_3) P_{Y_4|X_4}(y_4|x_4) \\ & \qquad \sum_{x_5,x_9,x_{13}} f_5 P_{Y_5|X_5}(y_5|x_5) P_{Y_9|X_9}(y_9|x_9) P_{Y_{13}|X_{13}}(y_{13}|x_{13}) \qquad (37) \\ = & \sum_{x_2,x_3,x_4,x_5,x_9,x_{13}} f_1 f_5 \prod_{i=1,2,3,4,5,9,13} P_{Y_i|X_i}(y_i|x_i) \end{aligned}$$

となる．図 21 に示す tree 構造における，x_1 から見て 1 段下の変数ノードからのメッセージを受け取り，因子ノードに従って $\sum$ の計算を行った式となっている．x_1 から見て 2 段下の変数ノードからのメッセージは受け取ることができていない．1 回目の計算で符号語が出力されなかった場合，2 回目の計算を行う．2 回目の計算を行った場合，左側の枝では，x_2 や x_3 の子ノードである，f_6, f_7 の因子ノードを通り，x_6 や x_7 等の 9 つの変数ノードからメッセージを受け取る．右側の枝では，x_5 や x_9 の子ノードである，f_2, f_3 の因子ノードを通り，x_6 や x_7 等の 9 つの変数ノードからメッセージを受け取る．したがって，x_6 や x_7 等の 9 つの変数ノードからのメッセージを重複して受け取ることになる．その様子は，図 23 の 2 回目の

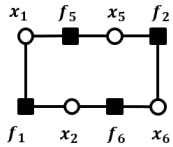
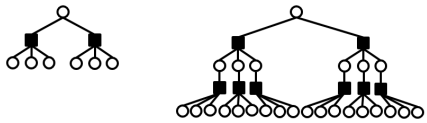


図 22　ラテン方陣のファクターグラフ内のループの例



計算1回目　　**計算2回目**

図 23　sum-product によるラテン方陣の周辺化計算値

sum-product による計算値の tree 構造が，図 21 の本来求めたかった tree 構造に対して，変数ノードが多くなっていることからわかる．2 回目の計算値により求まる計算値を記述する．ただし，図 23 において重複する変数ノードを区別するため右側の枝の 2 段目の変数ノードを x'_i とする．またそれに伴い x'_i をチェックする因子ノードも f'_i と記述し区別する．2 回目の計算値は，

$$
\begin{aligned}
&P_{Y_1|X_1}(y_1|x_1) \\
&\sum_{x_2,x_3,x_4} f_1 P_{Y_2|X_2}(y_2|x_2) \left(\sum_{x_6,x_{10},x_{14}} f_6 P_{Y_6|X_6}(y_6|x_6) P_{Y_{10}|X_{10}}(y_{10}|x_{10}) P_{Y_{14}|X_{14}}(y_{14}|x_{14}) \right) \\
&P_{Y_3|X_3}(y_3|x_3) \left(\sum_{x_7,x_{11},x_{15}} f_7 P_{Y_7|X_7}(y_7|x_7) P_{Y_{11}|X_{11}}(y_{11}|x_{11}) P_{Y_{15}|X_{15}}(y_{15}|x_{15}) \right) \\
&P_{Y_4|X_4}(y_4|x_4) \left(\sum_{x_8,x_{12},x_{16}} f_8 P_{Y_8|X_8}(y_8|x_8) P_{Y_{12}|X_{12}}(y_{12}|x_{12}) P_{Y_{16}|X_{16}}(y_{16}|x_{16}) \right) \\
&\sum_{x_5,x_9,x_{13}} f_5 P_{Y_5|X_5}(y_5|x_5) \left(\sum_{x'_6,x'_7,x'_8} f'_2 P_{Y'_6|X'_6}(y'_6|x'_6) P_{Y'_7|X'_7}(y'_7|x'_7) P_{Y'_8|X'_8}(y'_8|x'_8) \right) \\
&P_{Y_9|X_9}(y_9|x_9) \left(\sum_{x'_{10},x'_{11},x'_{12}} f'_3 P_{Y'_{10}|X'_{10}}(y'_{10}|x'_{10}) P_{Y'_{11}|X'_{11}}(y'_{11}|x'_{11}) P_{Y'_{12}|X'_{12}}(y'_{12}|x'_{12}) \right) \\
&P_{Y_{13}|X_{13}}(y_{13}|x_{13}) \left(\sum_{x'_{14},x'_{15},x'_{16}} f'_4 P_{Y'_{14}|X'_{14}}(y'_{14}|x'_{14}) P_{Y'_{15}|X'_{15}}(y'_{15}|x'_{15}) P_{Y'_{16}|X'_{16}}(y'_{16}|x'_{16}) \right) \\
=&\sum_{V\setminus x_1} \sum_{x'_6,x'_7,x'_8,x'_{10},x'_{11},x'_{12},x'_{14},x'_{15},x'_{16}} f_1 f'_2 f'_3 f'_4 f_5 f_6 f_7 f_8 \prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i) \\
&\prod_{j=6,7,8,10,11,12,14,15,16} P_{Y'_j|X'_j}(y'_j|x'_j)
\end{aligned}
\tag{38}
$$

となる．3 回目以降計算を行った場合も，図 23 と同様の傾向で tree の変数ノードが増えてい

く．本来の tree 構造は図 21 であるから，tree の深さより計算回数 2 回で全てのノードからのメッセージを受け取ることができる．しかし，sum-product アルゴリズムを行った場合，メッセージが重複してしまうため，求めたかった式 (36) の計算値と異なる値が出力される．これは，図 22 に示したラテン方陣のファクターグラフ内のループの影響である．

ラテン方陣では一つのマスを縦と横から 2 つのルールによってチェックを行っている．そのため，各変数ノードからは 2 本の枝が伸びている．図 22 のループが示すように，x_5 は f_5 を親ノードとし，f_2 を通して x_6 からのメッセージを受け取る．x_6 も同様に f_6 を通して x_2 からのメッセージを受け取ろうとする．x_2 も同様に f_1 を通して x_1 からのメッセージを受け取ろうとしてしまい，これを永遠に繰り返してしまうため計算値が収束しなくなってしまう．変数が送るメッセージは $P_{Y_i|X_i}(y_i|x_i)$ であり，因子ノード処理は $\sum$ の計算に対応しているので，計算回数が増え，tree の変数ノード，因子ノードが増えるとその分 $P_{Y_i|X_i}(y_i|x_i)$ の乗算と $\sum$ の計算が増えることがわかる．本来求めたい tree の変数ノードは 16 個であるが，sum-product アルゴリズムの 1 回目の計算では 7 個，2 回目の計算では 25 個になってしまう．3 回目以降の計算でも，変数ノードが増えることはあっても減ることは無いので，何回計算を繰り返しても正確な計算値を得ることができない．そのため，計算値が収束しないので，符号語が出力されなかった場合，永久に計算を繰り返してしまう可能性がある．ラテン方陣における sum-product では，図 24 に示すように計算のループ回数がある回数に達したら，ループを抜けてアルゴリズムを終わらせる必要がある．また，余分な変数ノードが増え $P_{Y_i|X_i}(y_i|x_i)$ が計算値に乗算されるため，$0 \leqq P_{Y_i|X_i}(y_i|x_i) \leqq 1$ の値が計算値に乗算されることになる．そのため，計算回数が増えるほど計算値が 0 に近づいて行ってしまうことが問題となる．配列で sum-product アルゴリズムの処理を行う場合，配列で処理できる値の範囲は決まっているので，計算値が限りなく小さくなってしまうと，単一の配列での値の処理ができなくなり，その問題を解決するために，新たなアルゴリズムを追加して計算量を増やさなければならないとい

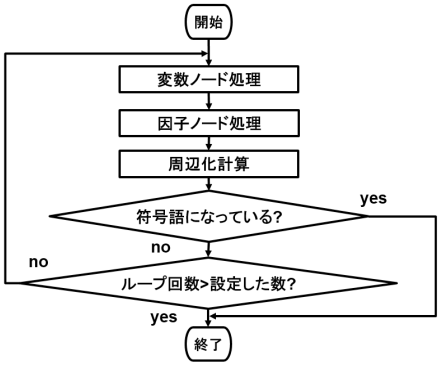


図 24　ラテン方陣における sum-product アルゴリズムのフローチャート

う問題が起こる．図 23 の tree 構造のように，因子ノードから見て子ノードは 3 つ存在するため，計算回数が増えるごとに追加される変数ノードの量は，前回の計算の tree の最終段の変数ノードの 3 倍となる．したがって，計算値に乗算される $0 \leqq P_{Y_i|X_i}(y_i|x_i) \leqq 1$ の値は指数関数的に増えていくことになる．

4　提案手法 1

3 章でラテン方陣に既存の復号方法を用いた場合に生じる問題について述べた．本章では，復号誤り確率と計算量の観点で，ラテン方陣の最適な復号方法について検討を行う．特に，実用化されている sum-product アルゴリズムに対して，ラテン方陣の tree 構造を活かして，ループによる性能の劣化への対策を考える．1.1 節で述べたように，ラテン方陣をそのまま通信として実用化することは難しい．しかし，ループによる性能の劣化はラテン方陣だけでなく一般的な符号でも生じる．ラテン方陣のループに対処した考え方が，他の符号の復号方法の考え方に応用できればよいと考えている．また，提案した復号手法についてシミュレーションによる評価を行う．

4.1　提案手法 1 の概要

3.2 節で述べたように，最尤復号法は計算量が多く，事前にすべてのラテン方陣を用意する必要がある．3.3 節で述べた，sum-product アルゴリズムは計算量を減らすことができ，ラテン方陣を用意しておく必要もない．しかし，ラテン方陣のようなファクターグラフ内にループを含む符号に sum-product アルゴリズムを適用すると，正確な周辺化計算値を求めることができない．本節では，ラテン方陣の符号の構造を活かして，sum-product アルゴリズムを改良することで，計算量を減らしつつ，復号誤り確率を低くすることができないか検討した．

以下，提案手法の概要を述べる．sum-product による計算の 2 回目の計算値式 (38) に着目すると，x_6, x_7, x_8, x_{10}, x_{11}, x_{12}, x_{14}, x_{15}, x_{16}, に関する値が重複している．そのため，それらに関する通信路の計算値が 2 乗で効いているように見える．i の値が同じであれば，$P_{Y_i|X_i}(y_i|x_i)$ と $P_{Y_i'|X_i'}(y_i'|x_i')$ は同じ値である．そのため，sum-product アルゴリズムの 1 回目の計算を $\sqrt{P_{Y_i|X_i}(y_i|x_i)}$ で行うことで，

$$\frac{\displaystyle\sum_{V\setminus x_1\ x_6',x_7',x_8',x_{10}',x_{11}',x_{12}',x_{14}',x_{15}',x_{16}'} f_1 f_2' f_3' f_4' f_5 f_6 f_7 f_8 \prod_{i=1,2,3,4,5,9,13} P_{Y_i|X_i}(y_i|x_i)}{\sqrt{\displaystyle\prod_{j=6,7,8,10,11,12,14,15,16} P_{Y_j|X_j}(y_j|x_j) \prod_{j=6,7,8,10,11,12,14,15,16} P_{Y_j'|X_j'}(y_j'|x_j')}} \tag{39}$$

を計算する．$\sum$ の回る範囲や f_2', f_3', f_4' が式 (36) と異なるため，完全に同じ値にすることはできないが，このように処理したほうが式 (38) よりも本来求めたい値である式 (36) に近い値

が得られるのではないかと考えた．また，sum-product のもともとのコンセプトは，ループのない符号については，計算を繰り返せば正確な計算値に収束し，元の tree 構造と同じ計算値が得られることであった．ループのある符号の場合，sum-product による計算値は収束せず，計算値にメッセージの値が乗算され続ける．しかし，計算回数が繰り返されることで，tree 構造に変数ノードが追加され，tree 構造の階層が増えて行ってしまうことは，本来の目的である元の tree 構造の計算値を求めたいという考えから遠ざかってしまうと考えた．ラテン方陣の tree 構造は図 21 であり，tree 構造の深さから計算回数 2 回で全ての変数ノードからの計算値を受け取ることができる．そのため，図 24 のように何回も計算のループを繰り返すのではなく，2 回だけ計算を繰り返し，それ以上計算を行わないアルゴリズムとする．さらに，x_1 だけでなく，全ての変数 x_i について図 21 と同じ tree 構造が得られる．x_i の tree 構造を描く際，求めたい変数ノードを一番上に描き，子ノードを下に描いていくが，図 25 に示すように，ラテン方陣の場合はどのマスを一番上に持ってきても，計算回数 2 回で全てのマスからのメッセージを受け取ることができる構造となっている．したがって，上述した議論は x_1 だけでなく，全ての x_i の周辺化計算値を求めるうえでも成立する．

図 26 に提案アルゴリズムのフローチャートを示す．最初の変数ノード処理で変数ノードが

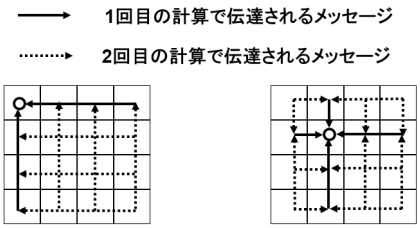


図 25　ラテン方陣の sum-product アルゴリズムによるメッセージの伝達の流れ

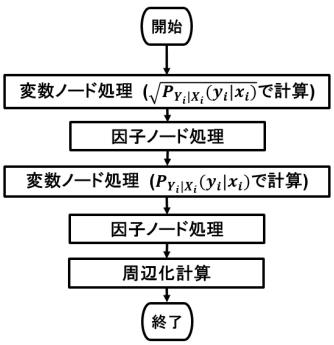


図 26　提案手法のフローチャート

送るメッセージを，$\sqrt{P_{Y_i|X_i}(y_i|x_i)}$ で計算し，2 回目の変数ノード処理では $P_{Y_i|X_i}(y_i|x_i)$ で計算する．計算の繰り返しは行わない．

4.2　シミュレーション結果と考察

図 26 のアルゴリズムと最尤復号法，sum-product アルゴリズムについて実際にシミュレーションを行い，復号誤り確率の比較を行った．シミュレーションは C 言語で行う．ラテン方陣を 576 個用意し符号語とする．符号語に対し，図 20 の通信路を通して，受信語を出力する．通信路の誤りは，メルセンヌ・ツイスタの乱数によって発生させる [8]．全ての符号語に対して 100 回ずつ通信路を通し，復号を行う．通信路の誤り確率は 0.01 ごとに測定する．復号誤り確率は，

$$\text{復号誤り確率} = \frac{\text{復号失敗と判定した数}}{\text{受信語の数}} \tag{40}$$

とする．復号失敗と判定するのは，符号語と復号語が一致しなかった場合，sum-product で設定した計算回数に達しても符号語が出力されない場合である．また，最尤復号法において，条件付確率が最大となる符号語が 2 つ存在する場合は復号失敗と判定する．これは，sum-produc アルゴリズムが，設定した計算回数までに復号できなかった場合に，適当に符号語を 1 個出力するプログラムになっていないため，条件をそろえるためである．使用したプログラムは付録 A.2 に記載する．

シミュレーション結果を図 27 に示す．横軸は通信路誤り確率である．縦軸は復号誤り確率であり，値が低いほど復号誤り確率が良い．凡例の sum-product は sum-product アルゴリズ

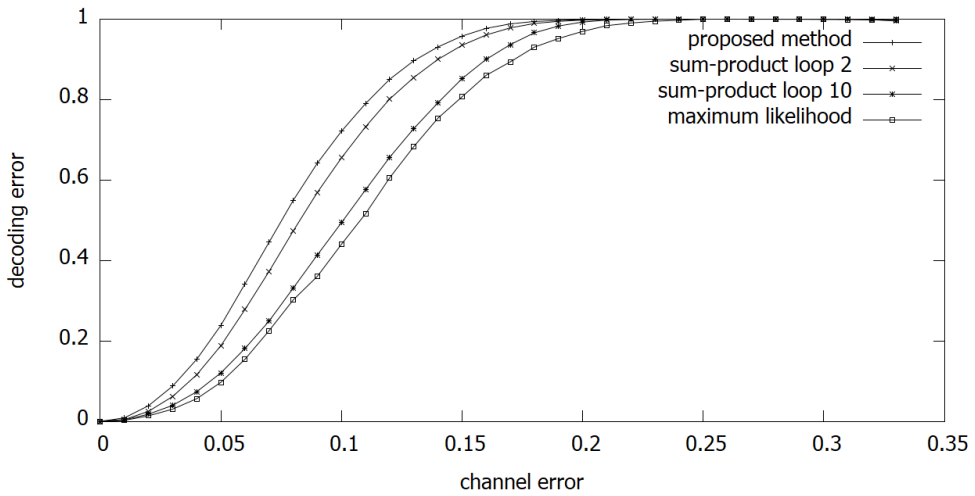


図 27　シミュレーション結果

ムであり，loop n は計算のループ回数の上限が n 回であることを表している．n 回目の計算を終えても復号できなかった場合，復号失敗と判定する．提案手法は，計算量では sum-product アルゴリズムの計算 2 回で終える場合と同じであるため，sum-product loop 2 の場合も測定した．ファクターグラフのループの影響で，sum-product アルゴリズムのほうが最尤復号法より，復号誤り確率が高いことがわかる．提案手法の復号誤り確率は，計算 2 回の sum-product アルゴリズムよりも低くなっている．計算量的には同じである計算 2 回の sum-product アルゴリズムよりも復号誤り確率が悪いので，式 (39) を用いた復号よりも，式 (38) を計算したほうが性能が良いことがわかる．図 23 の tree のように，sum-product アルゴリズムでは最終段の変数ノードからのメッセージが重複すると考え，送られてくるメッセージをルートにすることで，求めたかった本来の計算値に近づけたいと考えたが，性能をよくすることができなかった．また，sum-product アルゴリズムにおいて loop 10 のほうが loop 2 よりも復号誤り確率が良くなっている．ファクターグラフ内にループがある場合，計算が繰り返されるほど変数ノードが増え，本来求めたかった計算値と異なる値になると考えていたが，逆の結果が得られた．

5　提案手法 2

提案手法 1 では既存のアルゴリズムより性能を上げることができなかった．4.1 節より，1 回目の変数ノード処理をメッセージにルートをつけて行うという処理では性能が上がらなかった．また，sum-product の計算回数はループがある場合でも，計算回数を繰り返すほど復号誤り確率が低くなることがわかった．しかし，計算回数を増やすと，3.3 節で述べたように計算値が限りなく 0 に近づいて行ってしまうという問題がある．したがって，sum-product アルゴリズムを改善して，復号方法を考える場合，計算を繰り返して計算値が 0 に近づいていってしまうことを抑える方針か，4 章の復号方法のように，ラテン方陣では計算回数 2 回で全ての変数からメッセージを受け取ることができるという構造を活かし，変数ノード処理をルートで行うという処理ではなく，別の処理方法を考えて，周辺化計算値を本来求めたかった値である式 (36) に近づける方針のどちらかであると考えた．本章では，後述の計算回数を 2 回で終える方針で復号方法を考え提案する．

5.1　提案手法 2 の概要

式 (38) のように，sum-product アルゴリズムは，2 回目の計算で

$$\prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i) \prod_{j=6,7,8,10,11,12,14,15,16} P_{Y'_j|X'_j}(y'_j|x'_j) \tag{41}$$

の $\sum$ を取り計算を行う．本来求めたい値は，

$$\prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i) \tag{42}$$

の $\sum$ である．したがって，式 (41) の $\sum$ をいくらとっても，正しい計算値を求めることはできない．そのため，(41) 式の乗算されている値を減らすことで，正確な計算値により近い値を得られないか考えた．提案手法 2 では図 28 に示す tree 構造を計算する．図 28 の tree では 1 段目の右側の枝だけ，変数ノードが 1 段しかない tree となっている．つまり，右側の枝だけ 2 段目の変数ノードからメッセージが送られないことになる．この場合，計算値は，

$$\begin{aligned}
&P_{Y_1|X_1}(y_1|x_1) \\
&\sum_{x_2,x_3,x_4} f_1 P_{Y_2|X_2}(y_2|x_2) P_{Y_3|X_3}(y_3|x_3) P_{Y_4|X_4}(y_4|x_4) \\
&\sum_{x_5,x_9,x_{13}} f_5 P_{Y_5|X_5}(y_5|x_5) \left(\sum_{x_6,x_7,x_8} f_2 P_{Y_6|X_6}(y_6|x_6) P_{Y_7|X_7}(y_7|x_7) P_{Y_8|X_8}(y_8|x_8) \right) \\
&P_{Y_9|X_9}(y_9|x_9) \left(\sum_{x_{10},x_{11},x_{12}} f_3 P_{Y_{10}|X_{10}}(y_{10}|x_{10}) P_{Y_{11}|X_{11}}(y_{11}|x_{11}) P_{Y_{12}|X_{12}}(y_{12}|x_{12}) \right) \\
&P_{Y_{13}|X_{13}}(y_{13}|x_{13}) \left(\sum_{x_{14},x_{15},x_{16}} f_4 P_{Y_{14}|X_{14}}(y_{14}|x_{14}) P_{Y_{15}|X_{15}}(y_{15}|x_{15}) P_{Y_{16}|X_{16}}(y_{16}|x_{16}) \right) \\
=& \sum_{V \setminus x_1} f_1 f_2 f_3 f_4 f_5 \prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)
\end{aligned} \tag{43}$$

となる．つまり，式 (42) について f_6, f_7, f_8 のルールを取り除いて周辺化を行うという計算になる．式 (43) はそれらのルールを取り除いて $\sum$ を回す，つまり $\sum$ の中に f_6, f_7, f_8 を満

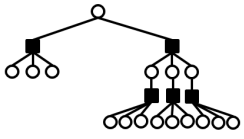


図 28　提案手法 2 による計算値

たさないものが含まれるということになる．したがって，

$$\begin{aligned}&\sum_{V\setminus x_1}\prod_{j=1}^{8} f_j \prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)\\ &+\sum_{V\setminus x_1} f_1f_2f_3f_4f_5\mathbb{1}[f_6,f_7,f_8\text{のルールを満たさない}]\prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)\end{aligned} \tag{44}$$

となり，第 1 項は正確な値であり，式 (36) の正確な周辺化計算値である．つまり，式 (43) は正確な計算値に，余分な計算値が足されている値になる．4.1 節の計算値は正確でない計算値 (41) を，正確でない範囲の $\sum$ で足す計算である．その計算よりも，正しい計算値に余分な値が足されている計算値のほうがより良い復号誤り確率が得られるのではないかと考え，式 (43) を計算することで復号を行い性能を検証した．

また，提案手法 2 の考え方を少し変えた，提案手法 3 についても検証を行った．提案手法 3 では図 29 に示す tree の値を計算する．まず，提案手法 2 と同様に右側の枝だけ 2 段目の変数ノードからメッセージを送った値を計算する．次に，今度は先ほどと逆に左側の枝だけ 2 段目の変数ノードを残した値を計算する．つまり，

$$\begin{aligned}&P_{Y_1|X_1}(y_1|x_1)\\ &\sum_{x_2,x_3,x_4} f_1P_{Y_2|X_2}(y_2|x_2)\left(\sum_{x_6,x_{10},x_{14}} f_6P_{Y_6|X_6}(y_6|x_6)P_{Y_{10}|X_{10}}(y_{10}|x_{10})P_{Y_{14}|X_{14}}(y_{14}|x_{14})\right)\\ &P_{Y_3|X_3}(y_3|x_3)\left(\sum_{x_7,x_{11},x_{15}} f_7P_{Y_7|X_7}(y_7|x_7)P_{Y_{11}|X_{11}}(y_{11}|x_{11})P_{Y_{15}|X_{15}}(y_{15}|x_{15})\right)\\ &P_{Y_4|X_4}(y_4|x_4)\left(\sum_{x_8,x_{12},x_{16}} f_8P_{Y_8|X_8}(y_8|x_8)P_{Y_{12}|X_{12}}(y_{12}|x_{12})P_{Y_{16}|X_{16}}(y_{16}|x_{16})\right)\\ &\sum_{x_5,x_9,x_{13}} f_5P_{Y_5|X_5}(y_5|x_5)P_{Y_9|X_9}(y_9|x_9)P_{Y_{13}|X_{13}}(y_{13}|x_{13})\\ &=\sum_{V\setminus x_1} f_1f_5f_6f_7f_8\prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)\end{aligned} \tag{45}$$

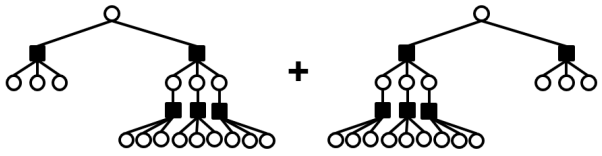


図 29　提案手法 3 による計算値

を計算する．この式は，式 (43) と同様に

$$\begin{aligned}&\sum_{V\setminus x_1}\prod_{j=1}^{8} f_j \prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)\\ &+\sum_{V\setminus x_1} f_1f_5f_6f_7f_8\mathbb{1}[f_2,f_3,f_4\text{のルールを満たさない}]\prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)\end{aligned} \tag{46}$$

と考えることができる．提案手法 3 では，式 (44) と式 (46) を足し合わせて，

$$\begin{aligned}&2\sum_{V\setminus x_1}\prod_{j=1}^{8} f_j \prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)\\ &+\sum_{V\setminus x_1} f_1f_5f_6f_7f_8\mathbb{1}[f_2,f_3,f_4\text{のルールを満たさない}]\prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)\\ &+\sum_{V\setminus x_1} f_1f_2f_3f_4f_5\mathbb{1}[f_6,f_7,f_8\text{のルールを満たさない}]\prod_{i=1}^{16} P_{Y_i|X_i}(y_i|x_i)\end{aligned} \tag{47}$$

を周辺化計算値と出力する．図 28 のように片側の枝だけ 2 段目の変数ノードを取り除いた場合，式 (43) のように正確な計算値に余分な値が足された値になる．逆側の枝の変数ノードを取り除いて計算した場合も同様であるが，取り除かれる因子ノードが異なるため，式 (45) で現れる余分な値は，式 (43) の余分な値とは異なる値になる．従って，式 (44) と式 (46) を足し合わせれば，余分な値は重複せず，正確な値のみ 2 倍にすることができ，正確な計算値の成分が強くなると考えた．

5.2 シミュレーション結果と考察

提案手法 2 と提案手法 3 についてもシミュレーションによる性能の比較を行った．シミュレーションの方法は 4.2 節と同じである．シミュレーション結果を図 30 に示す．図 30 より，提案手法 1 の結果から，計算値の処理方法を変えて復号を行ったが，sum-product より性能をよくすることはできなかった．ラテン方陣の符号の構造の，計算回数 2 回で全ての変数ノードからメッセージを受け取れる点を活かし，計算回数 2 回で復号誤り確率を下げられないかを検討してきた．1 回目の計算のメッセージをルートにする，片側の枝だけ 2 回目の計算で送るメッセージを止めるという処理を行ったが，どの提案手法も既存の sum-product アルゴリズムの計算回数 2 回の計算値による復号の性能を上回ることはできなかった．4 章と 5 章の提案手法では計算回数を 2 回で終える方針で復号方法を考えてきたが，現状では，計算回数 2 回で性能をよくする処理が提案できなかった．

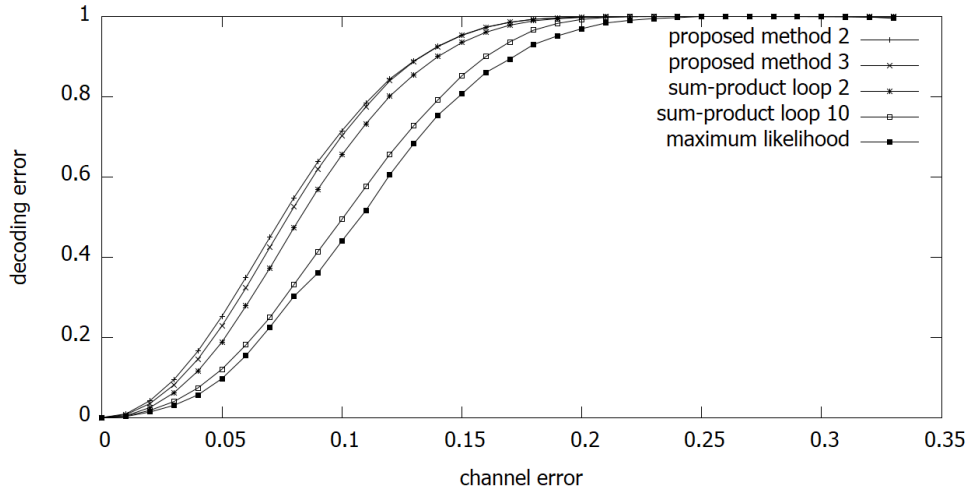


図 30　提案手法 2 と 3 のシミュレーション結果

6　sum-product の計算の繰り返し

4 章，5 章で述べたように，計算回数 2 回で処理を終える手法では復号誤り確率をよくすることができなかった．また，tree にループが存在する符号の場合，計算回数が増えるほど余分なノードが増え，正確な計算値に収束するという本来の目的から外れてしまうと考えたが，シミュレーション結果からは逆の傾向が得られた．本章ではラテン方陣の sum-product の計算の繰り返しについてより詳しく述べる．

図 31 にラテン方陣の sum-product の計算回数を 1 から 7 まで変えた場合の，通信路誤り確率に対する復号誤り確率をシミュレーションした結果を示す．シミュレーション方法は 4.2 節と同じである．図 31 からも計算回数が多いほど誤り確率が良くなっていることがわかる．また，計算回数 5 回以上になると，計算を繰り返しても復号誤り確率が良くならないことがわかる．これは，3.3 節で述べたように，計算を繰り返し続けると余分なノードが増えてしまい，その分通信路の値が計算値に乗算されて，計算値が限りなく小さくなってしまうからである．そのため，出力されるシンボルの値が 6 回目の計算以降は更新されず，5 回目の計算値で復号誤り確率が決まってしまう．

図 32 に，ある受信語に sum-product を適用した場合の，計算回数に対する $P_{X_1|Y}(x_1|\boldsymbol{y})$ の周辺化計算値を示す．横軸が計算回数である．縦軸が周辺化計算値の値であり，対数軸である．受信語は図 17 に示した例と同じ，最小の訂正回数のものが複数存在し，復号できないパターンのものである．そのため，計算を繰り返しても復号できず，計算を繰り返し続ける．計

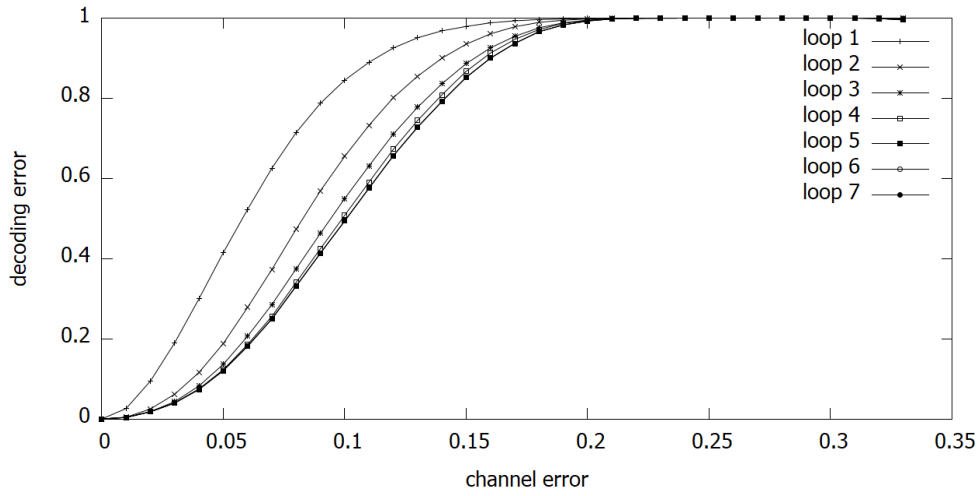


図 31　sum-product の計算回数を変えた場合のシミュレーション結果

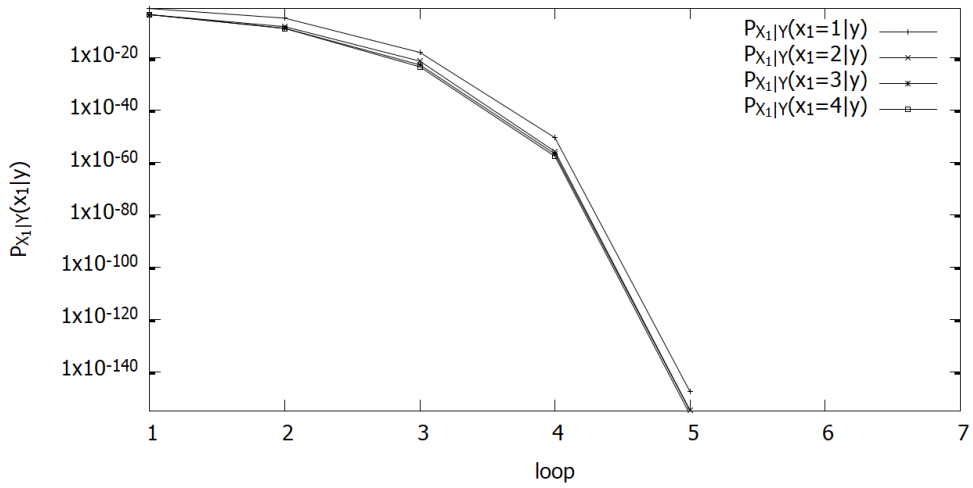


図 32　計算回数に対する x_1 の周辺化計算値

算値が 0 に近づいて行ってしまうのを確認するために，例としてそのような受信語を用い，x_1 の計算値のみに着目する．通信路誤り確率は 0.1 として，sum-product アルゴリズムを適用した．使用したプログラムは付録 A.3 に記載する．図 32 より，計算を繰り返すたびに計算値が小さくなることがわかる．6 回目の計算以降は，計算値は全て 0 である．また，x_1 の値がどの場合も 0 になってしまうため，計算値の比較ができず，復号ができなくなってしまうことがわかる．

図 33 に，図 32 のシミュレーションと同じ受信語で，sum-product を適用した場合の計算回数に対する $P_{X_1|Y}(x_1 = 1|\boldsymbol{y})$ の周辺化計算値を，通信路誤り確率 p を 0.1 から 0.00001 まで変化させてシミュレーションした結果を示す．シミュレーションの方法は図 32 のシミュレーションと同じであるが，図 33 では $x_1 = 1$ の周辺化計算値のみに着目し，通信路誤り確率 p を変化させてシミュレーションを行っている．図 33 から，通信路誤り確率が異なると計算値に近づく速さは異なるが，計算値が 0 に近づいて行ってしまうことがわかる．6 回目以降の計算値は全て 0 である．通信路誤り確率によらず，計算値が限りなく小さくなってしまう問題が発生することがわかる．

31 の結果が示すように，ラテン方陣では sum-product の計算回数が増えるほど，復号誤り確率が良くなった．符号の tree にループが存在しない場合，計算を繰り返していくと必ず正確な値に計算値が収束する．そのため，計算を繰り返すほど復号誤り確率がよくなると考えられる．しかし，tree にループが存在すると，本来求めたかった tree を求めることができない．計算を繰り返すほど余分なノードが増え，本来求めたかった tree の形と離れてしまい，復号誤り確率が悪くなると考えたが，図 31 では計算を繰り返すほど復号誤り確率が良くなった．ラテン方陣において, 計算回数を増やすほど復号誤り確率が良くなった理由として図 21 のように，tree 構造が非常に対称的になっていることと，図 25 で示したようにどのマスから見ても同じ tree 構造になることが考えられる．例として，図 34 のような tree に sum-product を適用した場合を考える．図 34 のような tree に sum-product を適用した場合，計算を繰り返していくと，左側の枝にループが存在するためその影響を受ける．2.3 節で述べたように，ループの影響は，tree に本来なかったノードが付加されることで現れる．その結果，図 34 のよ

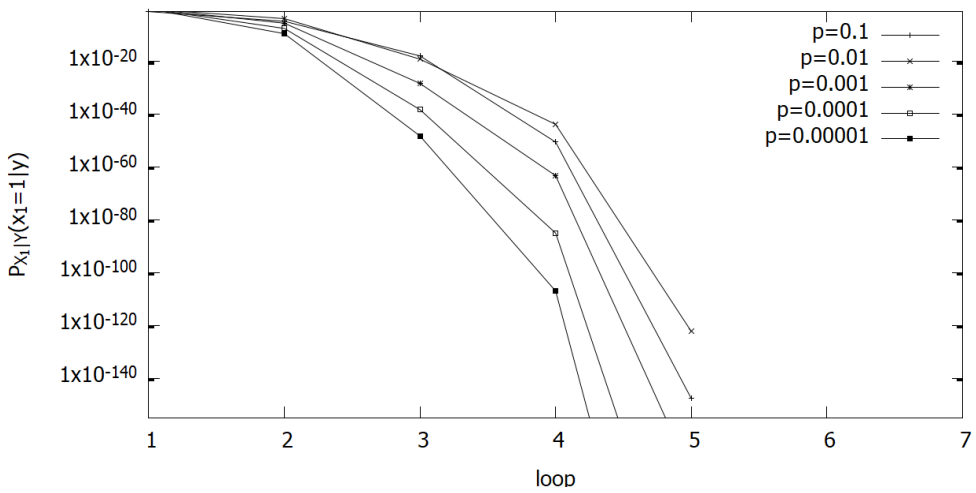


図 33　計算回数に対する $x_1 = 1$ の周辺化計算値

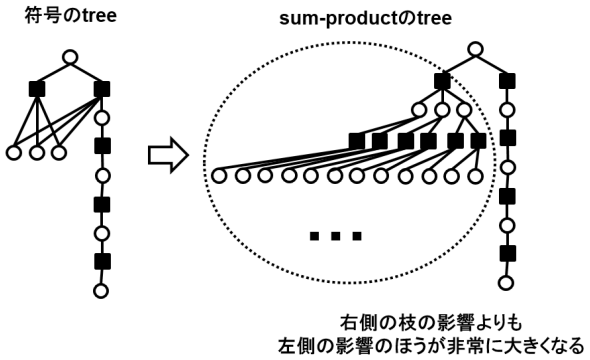


図 34　対称的でない tree の sum-product

うな符号の場合，右の枝の影響に対して，左側のループの影響が非常に大きくなってしまい，ループ部分の計算値で復号誤り確率が決まってしまうということが考えられる．そのため，単純に計算を繰り返しても，復号誤り確率が良くならないということが考えられる．それに対して，ラテン方陣の tree は図 21 のように非常に対称的になっており，計算を繰り返してループの影響を受けても，一部のループ部分のみの影響を受けることが無く，全てのマスからのメッセージをバランスよく受け取れるため，単純に計算を繰り返すほど復号誤り確率が良くなる傾向が得られたのではないかと考えられる．ループのある符号に sum-product アルゴリズムを適用した場合，ファクターグラフが疎であるほど良い復号誤り確率が得られることがわかっている．これは，2.3 節で述べたように，ループの影響によって余分なノードが付加されるため，ループの影響を減らすためには，tree 内のループの数を減らす，ループの長さを短くするといったことが有効だからであると考えられる．しかし，今回のラテン方陣における結果から，ループの影響を減らすための手法として，ラテン方陣のような対称的な tree 構造にするという手法もあるのではないかと考えられる．ただし，計算を繰り返して復号誤り確率が良くなった理由が，tree 構造が対称的であることのみに依存しているかどうかはわからない．例えば，ある tree 構造の符号に対して，tree 構造を変えて対称的なものにし，他の条件をそろえたうえで復号誤り確率を測定するというように，tree 構造が対称的であることの有効性を調べるといった事が必要であると考えられる．また，ループを許して対称的な構造にした場合，図 32，図 33 で示したループの影響で計算値が 0 に近づいて行ってしまう問題が起こるため，その対策が必要になる．

7　結論と今後の展望

7.1　結論

誤り訂正符号は通信の精度を高めるための技術である．誤り訂正の技術は送りたい情報をあるルールを用いて符号化することによって行う．ラテン方陣のような規則的なルールを誤り訂正符号に用いた場合に，最適な復号方法がどのようになるのか探索することで，新たな知見が得られないか検討した．最尤復号法によって復号を行った場合は，復号に必要な計算量が多く，最初にすべてのラテン方陣を用意しておく必要がある．sum-product アルゴリズムによって復号法を行った場合，計算量を減らすことができるが，計算値が収束せず，復号誤り確率が落ちてしまう．sum-product アルゴリズムを改良することで，計算量を減らしつつ，復号誤り確率を改善することができないかを検討した．ラテン方陣の tree 構造の，メッセージを計算 2 回で受け取ることができる，どのマスの周辺化関数を求める場合にも同じ tree 構造であるという点に着目しアルゴリズムの改善を狙った．いくつかの提案手法に対してシミュレーションによる検証を行ったが，sum-product アルゴリズムの性能を上回ることはできなかった．

7.2　今後の展望

ループのある符号に対し sum-product アルゴリズムを適用した場合，正確な計算値を求めることができない．本来の sum-product アルゴリズムのコンセプトは，ループのない符号であれば，計算を繰り返せば計算値が収束し，正確な値が求められるというものである．本研究では計算回数を 2 回で終えることで，本来求めたかった計算値に近づけることを狙ったが，良い結果が得られなかった．計算回数 2 回で得られる計算値に対して処理を行ってもよい結果が得られないのであれば，計算回数を繰り返す方針で復号したほうが良いということが考えられる．実際に，ラテン方陣に対して sum-product を適用した場合，計算回数を繰り返すほど良い結果が得られた．計算を繰り返すほど誤り確率が良くなった理由として，ラテン方陣の tree 構造が対称的であることが考えられる．今後の展望として，tree 構造が対称的であることの有効性をより詳しく調べるという方針が考えられる．しかし，ループのある符号で計算を繰り返すと計算値が急激に 0 に近づいていってしまうという問題がある．そのため，計算を繰り返すが，計算値が 0 に近づいてしまうことを抑えるようなアルゴリズムを提案する，といった方針が考えられる．また，sum-product アルゴリズムは，前回の計算で得られた計算値に対して，メッセージを送ることによって更新する漸化式のようなものだと考えられる．計算を sum-product で行うのではなく，繰り返すことによって収束するような計算式を考案することができれば新たな復号方法として利用できる，ということも方針として考えられる．

謝辞

本研究を行うにあたって，多くのご指導，研究環境をいただいた西新幹彦准教授に深く感謝いたします．また，日頃の研究室生活において，多くのご意見をいただいた西新研究室の方々に深く感謝いたします．

参考文献

[1] クロード・E. シャノン, ワレン・イーバー著; 植友彦訳, 通信の数学的理論, 筑摩書房, 2009.

[2] M. Nishiara and R. Hidai, “Decoding Error of Sudoku for Erasure Channels,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences,Vol. E100-A, NO. 12, pp. 2641-2646, December 2017.

[3] 数独の遊び方、ルール、解き方 — WEB ニコリ, https://www.nikoli.co.jp/ja/puzzles/sudoku/, 2020 年 12 月閲覧.

[4] 豊村聖也, ”ラテン方陣の消失通信路に対する復号誤り特性”, 信州大学工学部 卒業論文 (指導教員:西新幹彦), 2019 年 2 月.

[5] 和田山正, 誤り訂正技術の基礎, 森北出版, 2011.

[6] 芳沢光雄, 群論入門 対称性をはかる数学, 講談社, 2015.

[7] 岡田航, ”ラテン方陣の効率の良い数え上げに向けた基礎的考察”, 信州大学工学部 卒業論文 (指導教員:西新幹彦), 2018 年 3 月.

[8] Mersenne Twister: A random number generator (since 1997/10), http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/mt.html, 2020 年 12 月閲覧.

外部発表

[1] 塚田芳寿, 西新幹彦, “符号としてのラテン方陣に対する誤り訂正に向けて,” 電子情報通信学会信越支部大会, 1B-4, September 2019.

[2] 塚田芳寿, 西新幹彦, “誤り訂正符号としてのラテン方陣の復号方法について,” 電子情報通信学会信越支部大会, 1B-2, September 2019.

付録 A　ソースコード

A.1　4 行 4 列のラテン方陣を全て列挙するプログラム

```
#include <stdio.h>
#pragma warning(disable : 4996)

//ルール満たすかチェック
int latinrulecheck(int check[16])
{
    for (int i = 0; i < 4; i++) {
        int flag = 0;
        flag |= 1 << check[i + 0];
        flag |= 1 << check[i + 4];
        flag |= 1 << check[i + 8];
        flag |= 1 << check[i + 12];
        if (flag != 15) {
            return(-1);
        }
    }
    return(0);
}

//数字入れ替え
int main() {
    int permutation[24][4] = { {0,1,2,3},{0,1,3,2},{0,2,1,3},{0,2,3,1},{0,3,1,2},
{0,3,2,1},{1,0,2,3},{1,0,3,2},{1,2,0,3},{1,2,3,0},{1,3,0,2},{1,3,2,0},{2,0,1,3},
{2,0,3,1},{2,1,0,3},{2,1,3,0},{2,3,0,1},{2,3,1,0},{3,0,1,2},{3,0,2,1},{3,1,0,2}
,{3,1,2,0},{3,2,0,1},{3,2,1,0} };
    int latinsquare[576][16] = { { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }  };
    int check[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
    int sum = 0;

    for (int digit1 = 0; digit1 < 24; digit1++) {
        for (int digit2 = 0; digit2 < 24; digit2++) {
            for (int digit3 = 0; digit3 < 24; digit3++) {
                for (int digit4 = 0; digit4 < 24; digit4++) {
                    for (int i = 0; i < 4; i++) {
                        check[i + 0] = permutation[digit1][i];
                        check[i + 4] = permutation[digit2][i];
                        check[i + 8] = permutation[digit3][i];
                        check[i + 12] = permutation[digit4][i];
                    }
                    if (latinrulecheck(check) == 0) {
                        for (int i = 0; i < 16; i++) {
                            latinsquare[sum][i] = check[i];
                        }
                        sum++;
                    }
                }
            }
        }
    }
    for (int i = 0; i < 576; i++) {
        for (int k = 0; k < 16; k++) {
            printf("%d, ", latinsquare[i][k]);
        }
    }
}
```

A.2　復号方法の復号誤り確率の評価用プログラム

```
#include <stdio.h>
#include <math.h>
#pragma warning(disable : 4996)


//メルセンヌ・ツイスタの乱数生成 [8]
#define NN 312
#define MM 156
#define MATRIX_A 0xB5026F5AA96619E9ULL
#define UM 0xFFFFFFFF80000000ULL /* Most significant 33 bits */
#define LM 0x7FFFFFFFULL /* Least significant 31 bits */

/* The array for the state vector */
static unsigned long long mt[NN];
/* mti==NN+1 means mt[NN] is not initialized */
static int mti = NN + 1;

/* initializes mt[NN] with a seed */
void init_genrand64(unsigned long long seed)
{
    mt[0] = seed;
    for (mti = 1; mti < NN; mti++)
        mt[mti] = (6364136223846793005ULL * (mt[mti - 1] ^ (mt[mti - 1] >> 62)) + mti);
}

/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
void init_by_array64(unsigned long long init_key[],
    unsigned long long key_length)
{
    unsigned long long i, j, k;
    init_genrand64(19650218ULL);
    i = 1; j = 0;
    k = (NN > key_length ? NN : key_length);
    for (; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 62)) * 3935559000370003845ULL))
            + init_key[j] + j; /* non linear */
        i++; j++;
        if (i >= NN) { mt[0] = mt[NN - 1]; i = 1; }
        if (j >= key_length) j = 0;
    }
    for (k = NN - 1; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 62)) * 2862933555777941757ULL))
            - i; /* non linear */
        i++;
        if (i >= NN) { mt[0] = mt[NN - 1]; i = 1; }
    }

    mt[0] = 1ULL << 63; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0, 2^64-1]-interval */
unsigned long long genrand64_int64(void)
{
    int i;
    unsigned long long x;
    static unsigned long long mag01[2] = { 0ULL, MATRIX_A };

    if (mti >= NN) { /* generate NN words at one time */

        /* if init_genrand64() has not been called, */
        /* a default initial seed is used     */
```

```
        if (mti == NN + 1)
            init_genrand64(5489ULL);

        for (i = 0; i < NN - MM; i++) {
            x = (mt[i] & UM) | (mt[i + 1] & LM);
            mt[i] = mt[i + MM] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];
        }
        for (; i < NN - 1; i++) {
            x = (mt[i] & UM) | (mt[i + 1] & LM);
            mt[i] = mt[i + (MM - NN)] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];
        }
        x = (mt[NN - 1] & UM) | (mt[0] & LM);
        mt[NN - 1] = mt[MM - 1] ^ (x >> 1) ^ mag01[(int)(x & 1ULL)];

        mti = 0;
    }

    x = mt[mti++];

    x ^= (x >> 29) & 0x5555555555555555ULL;
    x ^= (x << 17) & 0x71D67FFFEDA60000ULL;
    x ^= (x << 37) & 0xFFF7EEE000000000ULL;
    x ^= (x >> 43);

    return x;
}

/* generates a random number on [0, 2^63-1]-interval */
long long genrand64_int63(void)
{
    return (long long)(genrand64_int64() >> 1);
}

/* generates a random number on [0,1]-real-interval */
double genrand64_real1(void)
{
    return (genrand64_int64() >> 11) * (1.0 / 9007199254740991.0);
}

/* generates a random number on [0,1)-real-interval */
double genrand64_real2(void)
{
    return (genrand64_int64() >> 11) * (1.0 / 9007199254740992.0);
}

/* generates a random number on (0,1)-real-interval */
double genrand64_real3(void)
{
    return ((genrand64_int64() >> 12) + 0.5) * (1.0 / 4503599627370496.0);
}


//通信路
void channel(int input[16], int output[16], double error) {
    double random = 0;
    for (int i = 0; i < 16; i++) {
        double random = genrand64_real2();
        if (random < error) {
            output[i] = (input[i] + 1) % 4;
        }
        else if (random < 2 * error) {
            output[i] = (input[i] + 2) % 4;
        }
        else if (random < 3 * error) {
            output[i] = (input[i] + 3) % 4;
        }
        else {
            output[i] = input[i];
        }
```

```
    }
    return;
}


//変数ノード処理
double function1(double Mxf[32][4], double error, int output[16], int Wx[32],
 double Mfx[32][4], int Mfxf[32])
{
    for (int i = 0; i < 32; i++) {
        for (int j = 0; j < 4; j++) {
            if (output[Wx[i]] != j) {
                Mxf[i][j] = error * Mfx[Mfxf[i]][j];
            }
            else {
                Mxf[i][j] = (1 - 3 * error) * Mfx[Mfxf[i]][j];
            }
        }
    }
    return 0;
}


//因子ノード処理
double function2(double Mxf[32][4], int Mxfx[32][3], int Perm[6][3], double Mfx[32][4])
{
    for (int i = 0; i < 32; i++) {
        for (int j = 0; j < 4; j++) {
            double Sum = 0;
            for (int l = 0; l < 6; l++) {
                double Prod = 1;
                Prod *= Mxf[Mxfx[i][0]][(Perm[l][0] + j) % 4]
* Mxf[Mxfx[i][1]][(Perm[l][1] + j) % 4]
 * Mxf[Mxfx[i][2]][(Perm[l][2] + j) % 4];
                Sum += Prod;
            }
            Mfx[i][j] = Sum;
        }
    }
    return 0;
}


//周辺化計算
double function3(double Mx[16][4], double error, int output[16], int Wx[32],
 double Mfx[32][4], int Mmar[16][2]) {
    for (int i = 0; i < 16; i++) {
        for (int j = 0; j < 4; j++) {
            if (output[Wx[i]] != j) {
                Mx[i][j] = error * Mfx[Mmar[i][0]][j] * Mfx[Mmar[i][1]][j];
            }
            else {
                Mx[i][j] = (1 - 3 * error) * Mfx[Mmar[i][0]][j] * Mfx[Mmar[i][1]][j];
            }
        }
    }
    return 0;
}


//符号語出力
int function4(double Mx[16][4], int Pmax[16]) {
    for (int i = 0; i < 16; i++) {
        for (int j = 0; j < 4; j++) {
            if (Mx[i][j] > Mx[i][Pmax[i]]) {
                Pmax[i] = j;
            }
        }
    }
```

```
    return 0;
}


//変数ノード処理 (ルートで計算)
double function5(double Mxf[32][4], double error, int output[16], int Wx[32],
 double Mfx[32][4], int Mfxf[32])
{
    for (int i = 0; i < 32; i++) {
        for (int j = 0; j < 4; j++) {
            if (output[Wx[i]] != j) {
                Mxf[i][j] = (sqrt(error)) * Mfx[Mfxf[i]][j];
            }
            else {
                Mxf[i][j] = (sqrt(1 - 3 * error)) * Mfx[Mfxf[i]][j];
            }
        }
    }
    return 0;
}


//変数ノード処理 (片側の枝のみ 1)
double function6(double Mxf[32][4], double error, int output[16], int Wx[32],
 double Mfx[32][4], int Mfxf[32])
{
    for (int i = 0; i < 32; i++) {
        for (int j = 0; j < 4; j++) {
            if (output[Wx[i]] != j) {
                if (i < 16) {
                    Mxf[i][j] = error * Mfx[Mfxf[i]][j];
                }
                else {
                    Mxf[i][j] = error;
                }
            }
            else {
                if (i < 16) {
                    Mxf[i][j] = (1 - 3 * error) * Mfx[Mfxf[i]][j];
                }
                else {
                    Mxf[i][j] = 1 - 3 * error;
                }
            }
        }
    }
    return 0;
}


//提案手法 3 の周辺化計算
double function7(double Mx[16][4], double error, int output[16], int Wx[32],
 double Mfx[32][4], int Mmar[16][2]) {
    for (int i = 0; i < 16; i++) {
        for (int j = 0; j < 4; j++) {
            if (output[Wx[i]] != j) {
                Mx[i][j] += error * Mfx[Mmar[i][0]][j] * Mfx[Mmar[i][1]][j];
            }
            else {
                Mx[i][j] += (1 - 3 * error) * Mfx[Mmar[i][0]][j] * Mfx[Mmar[i][1]][j];
            }
        }
    }
    return 0;
}


////変数ノード処理 (片側の枝のみ 2)
double function8(double Mxf[32][4], double error, int output[16], int Wx[32],
```

```
 double Mfx[32][4], int Mfxf[32])
{
    for (int i = 0; i < 32; i++) {
        for (int j = 0; j < 4; j++) {
            if (output[Wx[i]] != j) {
                if (i > 16) {
                    Mxf[i][j] = error * Mfx[Mfxf[i]][j];
                }
                else {
                    Mxf[i][j] = error;
                }
            }
            else {
                if (i > 16) {
                    Mxf[i][j] = (1 - 3 * error) * Mfx[Mfxf[i]][j];
                }
                else {
                    Mxf[i][j] = 1 - 3 * error;
                }
            }
        }
    }
    return 0;
}


//提案手法 3 での 2 回目の tree の計算値を求めるためにメッセージ初期化
double function9(double Mxf[32][4], double Mfx[32][4])
{
    for (int i = 0; i < 32; i++) {
        for (int j = 0; j < 4; j++) {
            Mxf[i][j] = 0;
        }
    }
    for (int i = 0; i < 32; i++) {
        for (int j = 0; j < 4; j++) {
            Mfx[i][j] = 1;
        }
    }
    return 0;
}


//最尤復号法
int decoding1(int input[16], int output[16], double error, int latinsquare[576][16]) {
    double probability[576] = { 0 };
    int Pmax = 0;
    for (int i = 0; i < 576; i++) {
        probability[i] = 1;
    }
    for (int i = 0; i < 576; i++) {
        for (int j = 0; j < 16; j++) {
            if (output[j] != latinsquare[i][j]) {
                probability[i] = probability[i] * error;
            }
            else {
                probability[i] = probability[i] * (1 - 3 * error);
            }
        }
    }
    for (int i = 0; i < 576; i++) {
        if (probability[Pmax] < probability[i]) {
            Pmax = i;
        }
    }
    for (int i = 0; i < 576; i++) {
        if (probability[Pmax] == probability[i]) {
            if (Pmax != i) {
                return (-1);
```

```
            }
        }
    }
    for (int i = 0; i < 16; i++) {
        if (input[i] != latinsquare[Pmax][i]) {
            return (-1);
        }
    }
    return (0);
}


//sum-roduct アルゴリズム loop10
double decoding2(int input[16], int output[16], double error) {
    double Mxf[32][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    double Mfx[32][4] = { {1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1} };
    double Mx[16][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0} };
    int Mfxf[32] = { 16,20,24,28,17,21,25,29,18,22,26,30,19,23,27,31,0,4,8,12,
1,5,9,13,2,6,10,14,3,7,11,15 };
    int Mxfx[32][3] = { {1,2,3},{0,2,3},{0,1,3},{0,1,2},{5,6,7},{4,6,7},{4,5,7},{4,5,6},
{9,10,11},{8,10,11},{8,9,11},{8,9,10},{13,14,15},{12,14,15},{12,13,15},{12,13,14},
{17,18,19},{16,18,19},{16,17,19},{16,17,18},{21,22,23,},{20,22,23},{20,21,23},
{20,21,22},{25,26,27},{24,26,27},{24,25,27},{24,25,26},{29,30,31},{28,30,31},
{28,29,31},{28,29,30} };
    int Wx[32] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,4,8,12,1,5,9,13,2,6,
10,14,3,7,11,15 };
    int Perm[6][3] = { {1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2,1} };
    int Mmar[16][2] = { {0,16},{1,20},{2,24},{3,28},{4,17},{5,21},{6,25},{7,29},
{8,18},{9,22},{10,26},{11,30},{12,19},{13,23},{14,27},{15,31} };
    int Deco[8][4] = { {0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15},{0,4,8,12},
{1,5,9,13},{2,6,10,14},{3,7,11,15} };
    int Calc = 0, Outp = 0;
    int Pmax[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
loop:function1(Mxf, error, output, Wx, Mfx, Mfxf);
     function2(Mxf, Mxfx, Perm, Mfx);
     function3(Mx, error, output, Wx, Mfx, Mmar);
     function4(Mx, Pmax);
    for (int i = 0; i < 8; i++) {
        int x = 0, y = 0, z = 0, w = 0;
        for (int j = 0; j < 4; j++) {
            if (Pmax[Deco[i][j]] == 0) {
                x++;
            }
            else if (Pmax[Deco[i][j]] == 1) {
                y++;
            }
            else if (Pmax[Deco[i][j]] == 2) {
                z++;
            }
            else {
                w++;
            }
        }
        if (x != 1 || y != 1 || z != 1 || w != 1) {
            Calc++;
            if (Calc < 10) {
                goto loop;
            }
            else {
                goto end;
            }
```

```
        }
    }
end:for (int i = 0; i < 16; i++) {
    if (input[i] != Pmax[i]) {
        return(-1);
    }
}
return(0);
}


//提案手法 1
double decoding3(int input[16], int output[16], double error) {
    double Mxf[32][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    double Mfx[32][4] = { {1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1} };
    double Mx[16][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    int Mfxf[32] = { 16,20,24,28,17,21,25,29,18,22,26,30,19,23,27,31,0,4,8,12,1,5,9,13,
2,6,10,14,3,7,11,15 };
    int Mxfx[32][3] = { {1,2,3},{0,2,3},{0,1,3},{0,1,2},{5,6,7},{4,6,7},{4,5,7},{4,5,6},{9,10,11},
{8,10,11},{8,9,11},{8,9,10},{13,14,15},{12,14,15},{12,13,15},{12,13,14},{17,18,19},{16,18,19},
{16,17,19},{16,17,18},{21,22,23,},{20,22,23},{20,21,23},{20,21,22},{25,26,27},{24,26,27},
{24,25,27},{24,25,26},{29,30,31},{28,30,31},{28,29,31},{28,29,30} };
    int Wx[32] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15 };
    int Perm[6][3] = { {1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2,1} };
    int Mmar[16][2] = { {0,16},{1,20},{2,24},{3,28},{4,17},{5,21},{6,25},{7,29},{8,18},{9,22},
{10,26},{11,30},{12,19},{13,23},{14,27},{15,31} };
    int Outp = 0;
    int Pmax[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
    function5(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function1(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function3(Mx, error, output, Wx, Mfx, Mmar);
    function4(Mx, Pmax);
    for (int i = 0; i < 16; i++) {
        if (input[i] != Pmax[i]) {
            return(-1);
        }
    }
    return(0);
}


//sum-product アルゴリズム  loop2
double decoding4(int input[16], int output[16], double error) {
    double Mxf[32][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    double Mfx[32][4] = { {1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1} };
    double Mx[16][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    int Mfxf[32] = { 16,20,24,28,17,21,25,29,18,22,26,30,19,23,27,31,0,4,8,12,1,5,9,13,
2,6,10,14,3,7,11,15 };
    int Mxfx[32][3] = { {1,2,3},{0,2,3},{0,1,3},{0,1,2},{5,6,7},{4,6,7},{4,5,7},{4,5,6},{9,10,11},
{8,10,11},{8,9,11},{8,9,10},{13,14,15},{12,14,15},{12,13,15},{12,13,14},{17,18,19},
{16,18,19},{16,17,19},{16,17,18},{21,22,23,},{20,22,23},{20,21,23},{20,21,22},{25,26,27},
{24,26,27},{24,25,27},{24,25,26},{29,30,31},{28,30,31},{28,29,31},{28,29,30} };
    int Wx[32] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15 };
```

```
    int Perm[6][3] = { {1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2,1} };
    int Mmar[16][2] = { {0,16},{1,20},{2,24},{3,28},{4,17},{5,21},{6,25},{7,29},{8,18},{9,22},
{10,26},{11,30},{12,19},{13,23},{14,27},{15,31} };
    int Deco[8][4] = { {0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15},{0,4,8,12},{1,5,9,13},
{2,6,10,14},{3,7,11,15} };
    int Calc = 0, Outp = 0;
    int Pmax[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
loop:function1(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function3(Mx, error, output, Wx, Mfx, Mmar);
    function4(Mx, Pmax);
    for (int i = 0; i < 8; i++) {
        int x = 0, y = 0, z = 0, w = 0;
        for (int j = 0; j < 4; j++) {
            if (Pmax[Deco[i][j]] == 0) {
                x++;
            }
            else if (Pmax[Deco[i][j]] == 1) {
                y++;
            }
            else if (Pmax[Deco[i][j]] == 2) {
                z++;
            }
            else {
                w++;
            }
        }
        if (x != 1 || y != 1 || z != 1 || w != 1) {
            Calc++;
            if (Calc < 2) {
                goto loop;
            }
            else {
                goto end;
            }
        }
    }
end:for (int i = 0; i < 16; i++) {
    if (input[i] != Pmax[i]) {
        return(-1);
    }
}
return(0);
}


//提案手法２
double decoding5(int input[16], int output[16], double error) {
    double Mxf[32][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    double Mfx[32][4] = { {1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1} };
    double Mx[16][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    int Mfxf[32] = { 16,20,24,28,17,21,25,29,18,22,26,30,19,23,27,31,0,4,8,12,1,5,9,13,
2,6,10,14,3,7,11,15 };
    int Mxfx[32][3] = { {1,2,3},{0,2,3},{0,1,3},{0,1,2},{5,6,7},{4,6,7},{4,5,7},{4,5,6},{9,10,11},
{8,10,11},{8,9,11},{8,9,10},{13,14,15},{12,14,15},{12,13,15},{12,13,14},{17,18,19},
{16,18,19},{16,17,19},{16,17,18},{21,22,23,},{20,22,23},{20,21,23},{20,21,22},
{25,26,27},{24,26,27},{24,25,27},{24,25,26},{29,30,31},{28,30,31},{28,29,31},{28,29,30} };
    int Wx[32] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15 };
    int Perm[6][3] = { {1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2,1} };
    int Mmar[16][2] = { {0,16},{1,20},{2,24},{3,28},{4,17},{5,21},{6,25},{7,29},{8,18},{9,22},
{10,26},{11,30},{12,19},{13,23},{14,27},{15,31} };
    int Outp = 0;
```

```
    int Pmax[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
    function1(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function6(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function3(Mx, error, output, Wx, Mfx, Mmar);
    function4(Mx, Pmax);
    for (int i = 0; i < 16; i++) {
        if (input[i] != Pmax[i]) {
            return(-1);
        }
    }
    return(0);
}


//提案手法 3
double decoding6(int input[16], int output[16], double error) {
    double Mxf[32][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    double Mfx[32][4] = { {1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1} };
    double Mx[16][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    int Mfxf[32] = { 16,20,24,28,17,21,25,29,18,22,26,30,19,23,27,31,0,4,8,12,1,5,9,13,
2,6,10,14,3,7,11,15 };
    int Mxfx[32][3] = { {1,2,3},{0,2,3},{0,1,3},{0,1,2},{5,6,7},{4,6,7},{4,5,7},{4,5,6},{9,10,11}
,{8,10,11},{8,9,11},{8,9,10},{13,14,15},{12,14,15},{12,13,15},{12,13,14},{17,18,19},
{16,18,19},{16,17,19},{16,17,18},{21,22,23,},{20,22,23},{20,21,23},{20,21,22},
{25,26,27},{24,26,27},{24,25,27},{24,25,26},{29,30,31},{28,30,31},{28,29,31},{28,29,30} };
    int Wx[32] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15 };
    int Perm[6][3] = { {1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2,1} };
    int Mmar[16][2] = { {0,16},{1,20},{2,24},{3,28},{4,17},{5,21},{6,25},{7,29},{8,18},{9,22},
{10,26},{11,30},{12,19},{13,23},{14,27},{15,31} };
    int Outp = 0;
    int Pmax[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
    function1(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function6(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function7(Mx, error, output, Wx, Mfx, Mmar);
    function9(Mxf, Mfx);
    function1(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function8(Mxf, error, output, Wx, Mfx, Mfxf);
    function2(Mxf, Mxfx, Perm, Mfx);
    function7(Mx, error, output, Wx, Mfx, Mmar);
    function4(Mx, Pmax);
    for (int i = 0; i < 16; i++) {
        if (input[i] != Pmax[i]) {
            return(-1);
        }
    }
    return(0);
}


//復号誤り確率測定
int main() {
    printf("error, rate1, rate2, rate3, rate3, rate4, rate5, rate6\n");
    int latinsquare[576][16] = { 0, 1, 2, 3, 1, 0, 3, 2, 2, 3, 0, 1, 3, 2, 1, 0, 0, 1, 2, 3, 1, 0,
 3, 2, 2, 3, 1, 0, 3, 2, 0, 1, 0, 1, 2, 3, 1, 0, 3, 2, 3, 2, 0, 1, 2, 3, 1, 0, 0, 1, 2, 3, 1, 0, 3, 2,
 3, 2, 1, 0, 2, 3, 0, 1, 0, 1, 2, 3, 1, 2, 3, 0, 2, 3, 0, 1, 3, 0, 1, 2, 0, 1, 2, 3, 1, 2, 3, 0, 3, 0,
 1, 2, 2, 3, 0, 1, 0, 1, 2, 3, 1, 3, 0, 2, 2, 0, 3, 1, 3, 2, 1, 0, 0, 1, 2, 3, 1, 3, 0, 2, 3, 2, 1, 0,
 2, 0, 3, 1, 0, 1, 2, 3, 2, 0, 3, 1, 1, 3, 0, 2, 3, 2, 1, 0, 0, 1, 2, 3, 2, 0, 3, 1, 3, 2, 1, 0, 1, 3,
```

0, 2, 0, 1, 2, 3, 2, 3, 0, 1, 1, 0, 3, 2, 3, 2, 1, 0, 0, 1, 2, 3, 2, 3, 0, 1, 1, 2, 3, 0, 3, 0, 1, 2,
0, 1, 2, 3, 2, 3, 0, 1, 3, 0, 1, 2, 1, 2, 3, 0, 0, 1, 2, 3, 2, 3, 0, 1, 3, 2, 1, 0, 1, 0, 3, 2, 0, 1,
2, 3, 2, 3, 1, 0, 1, 0, 3, 2, 3, 2, 0, 1, 0, 1, 2, 3, 2, 3, 1, 0, 3, 2, 0, 1, 1, 0, 3, 2, 0, 1, 2, 3,
3, 0, 1, 2, 1, 2, 3, 0, 2, 3, 0, 1, 0, 1, 2, 3, 3, 0, 1, 2, 2, 3, 0, 1, 1, 2, 3, 0, 0, 1, 2, 3, 3, 2,
0, 1, 1, 0, 3, 2, 2, 3, 1, 0, 0, 1, 2, 3, 3, 2, 0, 1, 2, 3, 1, 0, 1, 0, 3, 2, 0, 1, 2, 3, 3, 2, 1, 0,
1, 0, 3, 2, 2, 3, 0, 1, 0, 1, 2, 3, 3, 2, 1, 0, 1, 3, 0, 2, 2, 0, 3, 1, 0, 1, 2, 3, 3, 2, 1, 0, 2, 0,
3, 1, 1, 3, 0, 2, 0, 1, 2, 3, 3, 2, 1, 0, 2, 3, 0, 1, 1, 0, 3, 2, 0, 1, 3, 2, 1, 0, 2, 3, 2, 3, 0, 1,
3, 2, 1, 0, 0, 1, 3, 2, 1, 0, 2, 3, 2, 3, 1, 0, 3, 2, 0, 1, 0, 1, 3, 2, 1, 0, 2, 3, 3, 2, 0, 1, 2, 3,
1, 0, 0, 1, 3, 2, 1, 0, 2, 3, 3, 2, 1, 0, 2, 3, 0, 1, 0, 1, 3, 2, 1, 2, 0, 3, 2, 3, 1, 0, 3, 0, 2, 1,
0, 1, 3, 2, 1, 2, 0, 3, 3, 0, 2, 1, 2, 3, 1, 0, 0, 1, 3, 2, 1, 3, 2, 0, 2, 0, 1, 3, 3, 2, 0, 1, 0, 1,
3, 2, 1, 3, 2, 0, 3, 2, 0, 1, 2, 0, 1, 3, 0, 1, 3, 2, 2, 0, 1, 3, 1, 3, 2, 0, 3, 2, 0, 1, 0, 1, 3, 2,
2, 0, 1, 3, 3, 2, 0, 1, 1, 3, 2, 0, 0, 1, 3, 2, 2, 3, 0, 1, 1, 0, 2, 3, 3, 2, 1, 0, 0, 1, 3, 2, 2, 3,
0, 1, 3, 2, 1, 0, 1, 0, 2, 3, 0, 1, 3, 2, 2, 3, 1, 0, 1, 0, 2, 3, 3, 2, 0, 1, 0, 1, 3, 2, 2, 3, 1, 0,
1, 2, 0, 3, 3, 0, 2, 1, 0, 1, 3, 2, 2, 3, 1, 0, 3, 0, 2, 1, 1, 2, 0, 3, 0, 1, 3, 2, 2, 3, 1, 0, 3, 2,
0, 1, 1, 0, 2, 3, 0, 1, 3, 2, 3, 0, 2, 1, 1, 2, 0, 3, 2, 3, 1, 0, 0, 1, 3, 2, 3, 0, 2, 1, 2, 3, 1, 0,
1, 2, 0, 3, 0, 1, 3, 2, 3, 2, 0, 1, 1, 0, 2, 3, 2, 3, 1, 0, 0, 1, 3, 2, 3, 2, 0, 1, 1, 3, 2, 0, 2, 0,
1, 3, 0, 1, 3, 2, 3, 2, 0, 1, 2, 0, 1, 3, 1, 3, 2, 0, 0, 1, 3, 2, 3, 2, 0, 1, 2, 3, 1, 0, 1, 0, 2, 3,
0, 1, 3, 2, 3, 2, 1, 0, 1, 0, 2, 3, 2, 3, 0, 1, 0, 1, 3, 2, 3, 2, 1, 0, 2, 3, 0, 1, 1, 0, 2, 3, 0, 2,
1, 3, 1, 0, 3, 2, 2, 3, 0, 1, 3, 1, 2, 0, 0, 2, 1, 3, 1, 0, 3, 2, 3, 1, 2, 0, 2, 3, 0, 1, 0, 2, 1, 3,
1, 3, 0, 2, 2, 0, 3, 1, 3, 1, 2, 0, 0, 2, 1, 3, 1, 3, 0, 2, 2, 1, 3, 0, 3, 0, 2, 1, 0, 2, 1, 3, 1, 3,
0, 2, 3, 0, 2, 1, 2, 1, 3, 0, 0, 2, 1, 3, 1, 3, 0, 2, 3, 1, 2, 0, 2, 0, 3, 1, 0, 2, 1, 3, 1, 3, 2, 0,
2, 0, 3, 1, 3, 1, 0, 2, 0, 2, 1, 3, 1, 3, 2, 0, 3, 1, 0, 2, 2, 0, 3, 1, 0, 2, 1, 3, 2, 0, 3, 1, 1, 3,
0, 2, 3, 1, 2, 0, 0, 2, 1, 3, 2, 0, 3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 0, 2, 1, 3, 2, 0, 3, 1, 3, 1, 0, 2,
1, 3, 2, 0, 0, 2, 1, 3, 2, 0, 3, 1, 3, 1, 2, 0, 1, 3, 0, 2, 0, 2, 1, 3, 2, 1, 3, 0, 1, 3, 0, 2, 3, 0,
2, 1, 0, 2, 1, 3, 2, 1, 3, 0, 3, 0, 2, 1, 1, 3, 0, 2, 0, 2, 1, 3, 2, 3, 0, 1, 1, 0, 3, 2, 3, 1, 2, 0,
0, 2, 1, 3, 2, 3, 0, 1, 3, 1, 2, 0, 1, 0, 3, 2, 0, 2, 1, 3, 3, 0, 2, 1, 1, 3, 0, 2, 2, 1, 3, 0, 0, 2,
1, 3, 3, 0, 2, 1, 2, 1, 3, 0, 1, 3, 0, 2, 0, 2, 1, 3, 3, 1, 0, 2, 1, 3, 2, 0, 2, 0, 3, 1, 0, 2, 1, 3,
3, 1, 0, 2, 2, 0, 3, 1, 1, 3, 2, 0, 0, 2, 1, 3, 3, 1, 2, 0, 1, 0, 3, 2, 2, 3, 0, 1, 0, 2, 1, 3, 3, 1,
2, 0, 1, 3, 0, 2, 2, 0, 3, 1, 0, 2, 1, 3, 3, 1, 2, 0, 2, 0, 3, 1, 1, 3, 0, 2, 0, 2, 1, 3, 3, 1, 2, 0,
2, 3, 0, 1, 1, 0, 3, 2, 0, 2, 3, 1, 1, 0, 2, 3, 2, 3, 1, 0, 3, 1, 0, 2, 0, 2, 3, 1, 1, 0, 2, 3, 3, 1,
0, 2, 2, 3, 1, 0, 0, 2, 3, 1, 1, 3, 0, 2, 2, 0, 1, 3, 3, 1, 2, 0, 0, 2, 3, 1, 1, 3, 0, 2, 3, 1, 2, 0,
2, 0, 1, 3, 0, 2, 3, 1, 1, 3, 2, 0, 2, 0, 1, 3, 3, 1, 0, 2, 0, 2, 3, 1, 1, 3, 2, 0, 2, 1, 0, 3, 3, 0,
1, 2, 0, 2, 3, 1, 1, 3, 2, 0, 3, 0, 1, 2, 2, 1, 0, 3, 0, 2, 3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 2, 0, 1, 3,
0, 2, 3, 1, 2, 0, 1, 3, 1, 3, 0, 2, 3, 1, 2, 0, 0, 2, 3, 1, 2, 0, 1, 3, 1, 3, 2, 0, 3, 1, 0, 2, 1, 0,
3, 1, 2, 0, 1, 3, 3, 1, 0, 2, 1, 3, 2, 0, 0, 2, 3, 1, 2, 0, 1, 3, 3, 1, 2, 0, 1, 3, 0, 2, 0, 2, 3, 1,
2, 1, 0, 3, 1, 3, 2, 0, 3, 0, 1, 2, 0, 2, 3, 1, 2, 1, 0, 3, 3, 0, 1, 2, 1, 3, 2, 0, 0, 2, 3, 1, 2, 3,
1, 0, 1, 0, 2, 3, 3, 1, 0, 2, 0, 2, 3, 1, 2, 3, 1, 0, 3, 1, 0, 2, 1, 0, 2, 3, 0, 2, 3, 1, 3, 0, 1, 2,
1, 3, 2, 0, 2, 1, 0, 3, 0, 2, 3, 1, 3, 0, 1, 2, 2, 1, 0, 3, 1, 3, 2, 0, 0, 2, 3, 1, 3, 1, 0, 2, 1, 0,
2, 3, 2, 3, 1, 0, 0, 2, 3, 1, 3, 1, 0, 2, 1, 3, 2, 0, 2, 0, 1, 3, 0, 2, 3, 1, 3, 1, 0, 2, 2, 0, 1, 3,
1, 3, 2, 0, 0, 2, 3, 1, 3, 1, 0, 2, 2, 3, 1, 0, 1, 0, 2, 3, 0, 2, 3, 1, 3, 1, 2, 0, 1, 3, 0, 2, 2, 0,
1, 3, 0, 2, 3, 1, 3, 1, 2, 0, 2, 0, 1, 3, 0, 2, 0, 3, 1, 2, 1, 0, 2, 3, 2, 1, 3, 0, 3, 2, 0, 1,
0, 3, 1, 2, 1, 0, 2, 3, 3, 2, 0, 1, 2, 1, 3, 0, 0, 3, 1, 2, 1, 2, 0, 3, 2, 0, 3, 1, 3, 1, 2, 0, 0, 3,
1, 2, 1, 2, 0, 3, 2, 1, 3, 0, 3, 0, 2, 1, 0, 3, 1, 2, 1, 2, 0, 3, 3, 0, 2, 1, 2, 1, 3, 0, 0, 3, 1, 2,
1, 2, 0, 3, 3, 1, 2, 0, 2, 0, 3, 1, 0, 3, 1, 2, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 2, 1, 0, 3, 1, 2, 1, 2,
3, 0, 3, 0, 2, 1, 2, 1, 0, 3, 0, 3, 1, 2, 2, 0, 3, 1, 1, 2, 0, 3, 3, 1, 2, 0, 0, 3, 1, 2, 2, 0, 3, 1,
3, 1, 2, 0, 1, 2, 0, 3, 0, 3, 1, 2, 2, 1, 0, 3, 1, 2, 3, 0, 3, 0, 2, 1, 0, 3, 1, 2, 2, 1, 0, 3, 3, 0,
2, 1, 1, 2, 3, 0, 0, 3, 1, 2, 2, 1, 3, 0, 1, 0, 2, 3, 3, 2, 0, 1, 0, 3, 1, 2, 2, 1, 3, 0, 1, 2, 0, 3,
3, 0, 2, 1, 0, 3, 1, 2, 2, 1, 3, 0, 3, 0, 2, 1, 1, 2, 0, 3, 0, 3, 1, 2, 2, 1, 3, 0, 3, 2, 0, 1, 1, 0,
2, 3, 0, 3, 1, 2, 3, 0, 2, 1, 1, 2, 0, 3, 2, 1, 3, 0, 0, 3, 1, 2, 3, 0, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3,
0, 3, 1, 2, 3, 0, 2, 1, 2, 1, 0, 3, 1, 2, 3, 0, 0, 3, 1, 2, 3, 0, 2, 1, 2, 1, 3, 0, 1, 2, 0, 3, 0, 3,
1, 2, 3, 1, 2, 0, 1, 2, 0, 3, 2, 0, 3, 1, 0, 3, 1, 2, 3, 1, 2, 0, 2, 0, 3, 1, 1, 2, 0, 3, 0, 3, 1, 2,
3, 2, 0, 1, 1, 0, 2, 3, 2, 1, 3, 0, 0, 3, 1, 2, 3, 2, 0, 1, 2, 1, 3, 0, 1, 0, 2, 3, 0, 3, 2, 1, 1, 0,
3, 2, 2, 1, 0, 3, 3, 2, 1, 0, 0, 3, 2, 1, 1, 0, 3, 2, 3, 2, 1, 0, 2, 1, 0, 3, 0, 3, 2, 1, 1, 2, 0, 3,
2, 1, 3, 0, 3, 0, 1, 2, 0, 3, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 1, 2, 2, 1, 3, 0, 0, 3, 2, 1, 1, 2,
3, 0, 2, 0, 1, 2, 3, 0, 3, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 1, 2, 2, 1, 3, 0, 0, 3, 2, 1, 1, 2, 3,
2, 1, 0, 3, 0, 3, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 1, 2, 2, 1, 3, 0, 0, 3, 2, 1, 1, 2, 3, 0, 2, 0,
1, 3, 3, 1, 0, 2, 0, 3, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 1, 2, 2, 1, 3, 0, 0, 3, 2, 1, 1, 2, 3, 0,
2, 1, 0, 3, 0, 3, 2, 1, 1, 2, 3, 0, 3, 1, 0, 2, 2, 0, 1, 3, 0, 3, 2, 1, 2, 0, 1, 3, 1, 2, 3, 0, 3, 1,
0, 2, 0, 3, 2, 1, 2, 0, 1, 3, 3, 1, 0, 2, 1, 2, 3, 0, 0, 3, 2, 1, 2, 1, 0, 3, 1, 0, 3, 2, 3, 2, 1, 0,
0, 3, 2, 1, 2, 1, 0, 3, 1, 2, 3, 0, 3, 0, 1, 2, 0, 3, 2, 1, 2, 1, 0, 3, 3, 0, 1, 2, 1, 2, 3, 0, 0, 3,
2, 1, 2, 1, 0, 3, 3, 2, 1, 0, 1, 0, 3, 2, 0, 3, 2, 1, 2, 1, 3, 0, 1, 2, 0, 3, 3, 0, 1, 2, 0, 3, 2, 1,
2, 1, 3, 0, 3, 0, 1, 2, 1, 2, 0, 3, 0, 3, 2, 1, 3, 0, 1, 2, 1, 2, 0, 3, 2, 1, 3, 0, 0, 3, 2, 1, 3, 0,
1, 2, 1, 2, 3, 0, 2, 1, 0, 3, 0, 3, 2, 1, 3, 0, 1, 2, 2, 1, 0, 3, 1, 2, 3, 0, 0, 3, 2, 1, 3, 0, 1, 2,
2, 1, 3, 0, 1, 2, 0, 3, 0, 3, 2, 1, 3, 1, 0, 2, 1, 2, 3, 0, 2, 0, 1, 3, 0, 3, 2, 1, 3, 1, 0, 2, 2, 0,
1, 3, 1, 2, 3, 0, 0, 3, 2, 1, 3, 2, 1, 0, 1, 0, 3, 2, 2, 1, 0, 3, 0, 3, 2, 1, 3, 2, 1, 0, 2, 0, 3, 1,
1, 0, 3, 2, 1, 0, 2, 3, 0, 1, 3, 2, 2, 3, 0, 1, 3, 2, 1, 0, 1, 0, 2, 3, 0, 1, 3, 2, 2, 3, 1, 0, 3, 2,
0, 1, 1, 0, 2, 3, 0, 1, 3, 2, 3, 2, 0, 1, 2, 3, 1, 0, 1, 0, 2, 3, 0, 1, 3, 2, 3, 2, 1, 0, 2, 3, 0, 1,
1, 0, 2, 3, 0, 2, 3, 1, 2, 3, 1, 0, 3, 1, 0, 2, 1, 0, 2, 3, 0, 2, 3, 1, 3, 1, 0, 2, 2, 3, 1, 0, 1, 0,
2, 3, 0, 3, 1, 2, 2, 1, 3, 0, 3, 2, 0, 1, 1, 0, 2, 3, 0, 3, 1, 2, 3, 2, 0, 1, 2, 1, 3, 0, 1, 2, 3, 2,
2, 1, 3, 0, 0, 3, 1, 2, 3, 2, 0, 1, 1, 0, 2, 3, 2, 1, 3, 0, 3, 2, 0, 1, 0, 2, 3, 1, 3, 2, 2, 3, 0, 1,
0, 1, 0, 1, 3, 2, 3, 2, 1, 0, 1, 0, 2, 3, 2, 3, 0, 1, 3, 2, 1, 0, 0, 1, 3, 2, 1, 0, 2, 3, 2, 3, 1, 0,

0, 1, 3, 2, 3, 2, 0, 1, 1, 0, 2, 3, 2, 3, 1, 0, 0, 2, 3, 1, 3, 1, 0, 2, 1, 0, 2, 3, 2, 3, 1, 0, 3, 1,
0, 2, 0, 2, 3, 1, 1, 0, 2, 3, 2, 3, 1, 0, 3, 2, 0, 1, 0, 1, 3, 2, 1, 0, 2, 3, 3, 1, 0, 2, 0, 2, 3, 1,
2, 3, 1, 0, 1, 0, 2, 3, 3, 1, 0, 2, 2, 3, 1, 0, 0, 2, 3, 1, 1, 0, 2, 3, 3, 2, 0, 1, 0, 1, 3, 2, 2, 3,
1, 0, 1, 0, 2, 3, 3, 2, 0, 1, 0, 3, 1, 2, 2, 1, 3, 0, 1, 0, 2, 3, 3, 2, 0, 1, 2, 1, 3, 0, 0, 3, 1, 2,
1, 0, 2, 3, 3, 2, 0, 1, 2, 3, 1, 0, 0, 1, 3, 2, 1, 0, 2, 3, 3, 2, 1, 0, 0, 1, 3, 2, 2, 3, 0, 1, 1, 0,
2, 3, 3, 2, 1, 0, 2, 3, 0, 1, 0, 1, 3, 2, 1, 0, 3, 2, 0, 1, 2, 3, 2, 3, 0, 1, 3, 2, 1, 0, 1, 0, 3, 2,
0, 1, 2, 3, 2, 3, 1, 0, 3, 2, 0, 1, 1, 0, 3, 2, 0, 1, 2, 3, 3, 2, 0, 1, 2, 3, 1, 0, 1, 0, 3, 2, 0, 1,
2, 3, 3, 2, 1, 0, 2, 3, 0, 1, 1, 0, 3, 2, 0, 2, 1, 3, 2, 3, 0, 1, 3, 1, 2, 0, 1, 0, 3, 2, 0, 2, 1, 3,
3, 1, 2, 0, 2, 3, 0, 1, 1, 0, 3, 2, 0, 3, 2, 1, 2, 1, 0, 3, 3, 2, 1, 0, 1, 0, 3, 2, 0, 3, 2, 1, 3, 2,
1, 0, 2, 1, 0, 3, 1, 0, 3, 2, 2, 1, 0, 3, 0, 3, 2, 1, 3, 2, 1, 0, 1, 0, 3, 2, 2, 1, 0, 3, 3, 2, 1, 0,
0, 3, 2, 1, 1, 0, 3, 2, 2, 3, 0, 1, 0, 1, 2, 3, 3, 2, 1, 0, 1, 0, 3, 2, 2, 3, 0, 1, 0, 2, 1, 3, 3, 1,
2, 0, 1, 0, 3, 2, 2, 3, 0, 1, 3, 1, 2, 0, 0, 2, 1, 3, 1, 0, 3, 2, 2, 3, 0, 1, 3, 2, 1, 0, 0, 1, 2, 3,
1, 0, 3, 2, 2, 3, 1, 0, 0, 1, 2, 3, 3, 2, 0, 1, 1, 0, 3, 2, 2, 3, 1, 0, 3, 2, 0, 1, 0, 1, 2, 3, 1, 0,
3, 2, 3, 1, 2, 0, 0, 2, 1, 3, 2, 3, 0, 1, 1, 0, 3, 2, 3, 1, 2, 0, 2, 3, 0, 1, 0, 2, 1, 3, 1, 0, 3, 2,
3, 2, 0, 1, 0, 1, 2, 3, 2, 3, 1, 0, 1, 0, 3, 2, 3, 2, 0, 1, 2, 3, 1, 0, 0, 1, 2, 3, 1, 0, 3, 2, 3, 2,
1, 0, 0, 1, 2, 3, 2, 3, 0, 1, 1, 0, 3, 2, 3, 2, 1, 0, 0, 3, 2, 1, 2, 1, 0, 3, 1, 0, 3, 2, 3, 2, 1, 0,
2, 1, 0, 3, 0, 3, 2, 1, 1, 0, 3, 2, 3, 2, 1, 0, 2, 3, 0, 1, 0, 1, 2, 3, 1, 2, 0, 3, 0, 1, 3, 2, 2, 3,
1, 0, 3, 0, 2, 1, 1, 2, 0, 3, 0, 1, 3, 2, 3, 0, 2, 1, 2, 3, 1, 0, 1, 2, 0, 3, 0, 3, 1, 2, 2, 0, 3, 1,
3, 1, 2, 0, 1, 2, 0, 3, 0, 3, 1, 2, 2, 1, 3, 0, 3, 0, 2, 1, 1, 2, 0, 3, 0, 3, 1, 2, 3, 0, 2, 1, 2, 1,
3, 0, 1, 2, 0, 3, 0, 3, 2, 1, 3, 0, 1, 2, 2, 1, 3, 0, 1, 2, 0, 3, 2, 0, 3, 1, 0, 3, 1, 2, 3, 0, 1, 2,
1, 2, 0, 3, 0, 3, 2, 1, 3, 0, 1, 2, 2, 1, 3, 0, 1, 2, 0, 3, 2, 1, 3, 0, 3, 1, 0, 3, 1, 2, 3, 0, 1, 2,
0, 3, 2, 0, 3, 1, 3, 1, 2, 0, 0, 3, 1, 2, 1, 2, 0, 3, 2, 1, 3, 0, 0, 3, 1, 2, 3, 0, 2, 1, 1, 2, 0, 3,
2, 1, 3, 0, 0, 3, 2, 1, 3, 0, 1, 2, 1, 2, 0, 3, 2, 1, 3, 0, 3, 0, 1, 2, 0, 3, 2, 1, 1, 2, 0, 3, 2, 1,
3, 0, 3, 0, 2, 1, 0, 3, 1, 2, 1, 2, 0, 3, 2, 3, 1, 0, 0, 1, 3, 2, 3, 0, 2, 1, 1, 2, 0, 3, 2, 3, 1, 0,
3, 0, 2, 1, 0, 1, 3, 2, 1, 2, 0, 3, 3, 0, 1, 2, 0, 3, 2, 1, 2, 1, 3, 0, 1, 2, 0, 3, 3, 0, 1, 2, 2, 1,
3, 0, 0, 3, 2, 1, 1, 2, 0, 3, 3, 0, 2, 1, 0, 1, 3, 2, 2, 3, 1, 0, 1, 2, 0, 3, 3, 0, 2, 1, 0, 3, 1, 2,
2, 1, 3, 0, 1, 2, 0, 3, 3, 0, 2, 1, 2, 1, 3, 0, 0, 3, 1, 2, 1, 2, 0, 3, 3, 0, 2, 1, 2, 3, 1, 0, 0, 1,
3, 2, 1, 2, 0, 3, 3, 1, 2, 0, 0, 3, 1, 2, 2, 0, 3, 1, 1, 2, 0, 3, 3, 1, 2, 0, 2, 0, 3, 1, 0, 3, 1, 2,
1, 2, 3, 0, 0, 1, 2, 3, 2, 3, 0, 1, 3, 0, 1, 2, 1, 2, 3, 0, 0, 1, 2, 3, 3, 0, 1, 2, 2, 3, 0, 1, 1, 2,
3, 0, 0, 3, 1, 2, 2, 1, 0, 3, 3, 0, 2, 1, 0, 1, 3, 2, 2, 3, 1, 0, 1, 2, 0, 3, 3, 0, 2, 1, 2, 3, 0, 2,
0, 3, 2, 1, 2, 0, 1, 3, 3, 1, 0, 2, 1, 2, 3, 0, 0, 3, 2, 1, 2, 1, 0, 3, 3, 0, 1, 2, 1, 2, 3, 0, 0, 3,
2, 1, 3, 0, 1, 2, 2, 1, 0, 3, 1, 2, 3, 0, 0, 3, 2, 1, 3, 1, 0, 2, 2, 0, 1, 3, 1, 2, 3, 0, 2, 0, 1, 3,
0, 3, 2, 1, 3, 1, 0, 2, 1, 2, 3, 0, 2, 0, 1, 3, 3, 1, 0, 2, 0, 3, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 0, 3,
1, 2, 3, 0, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 0, 3, 2, 1, 3, 0, 1, 2, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 1, 2,
0, 3, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 2, 1, 0, 3, 1, 2, 1, 2, 3, 0, 2, 3, 0, 1, 0, 1, 2, 3, 3, 0,
1, 2, 1, 2, 3, 0, 2, 3, 0, 1, 3, 0, 1, 2, 0, 1, 2, 3, 1, 2, 3, 0, 3, 0, 1, 2, 0, 1, 2, 3, 2, 3, 0, 1,
1, 2, 3, 0, 3, 0, 1, 2, 0, 3, 2, 1, 2, 1, 0, 3, 1, 2, 3, 0, 3, 0, 1, 2, 2, 1, 0, 3, 0, 3, 2, 1, 1, 2,
3, 0, 3, 0, 1, 2, 2, 3, 0, 1, 0, 1, 2, 3, 1, 2, 3, 0, 3, 0, 2, 1, 0, 3, 1, 2, 2, 1, 0, 3, 1, 2, 3, 0,
3, 0, 2, 1, 2, 1, 0, 3, 0, 3, 1, 2, 1, 2, 3, 0, 3, 1, 0, 2, 0, 3, 2, 1, 2, 0, 1, 3, 1, 2, 3, 0, 3, 1,
0, 2, 2, 0, 1, 3, 0, 3, 2, 1, 1, 3, 0, 2, 0, 1, 2, 3, 2, 0, 3, 1, 3, 2, 1, 0, 1, 3, 0, 2, 0, 1, 2, 3,
3, 2, 1, 0, 2, 0, 3, 1, 1, 3, 0, 2, 0, 2, 1, 3, 2, 0, 3, 1, 3, 1, 2, 0, 1, 3, 0, 2, 0, 2, 1, 3, 2, 1,
3, 0, 3, 0, 2, 1, 1, 3, 0, 2, 0, 2, 1, 3, 3, 0, 2, 1, 2, 1, 3, 0, 1, 3, 0, 2, 0, 2, 1, 3, 3, 1, 2, 0,
2, 0, 3, 1, 1, 3, 0, 2, 0, 2, 3, 1, 2, 0, 1, 3, 3, 1, 2, 0, 1, 3, 0, 2, 0, 2, 3, 1, 3, 1, 2, 0, 2, 0,
1, 3, 1, 3, 0, 2, 2, 0, 1, 3, 0, 2, 3, 1, 3, 1, 2, 0, 1, 3, 0, 2, 2, 0, 1, 3, 3, 1, 2, 0, 0, 2, 3, 1,
1, 3, 0, 2, 2, 0, 3, 1, 0, 1, 2, 3, 3, 2, 1, 0, 1, 3, 0, 2, 2, 0, 3, 1, 0, 2, 1, 3, 3, 1, 2, 0, 1, 3,
0, 2, 2, 0, 3, 1, 3, 1, 2, 0, 0, 2, 1, 3, 1, 3, 0, 2, 2, 0, 3, 1, 3, 2, 1, 0, 0, 1, 2, 3, 1, 3, 0, 2,
2, 1, 3, 0, 0, 2, 1, 3, 3, 0, 2, 1, 1, 3, 0, 2, 2, 1, 3, 0, 3, 0, 2, 1, 0, 2, 1, 3, 1, 3, 0, 2, 3, 0,
2, 1, 0, 2, 1, 3, 2, 1, 3, 0, 1, 3, 0, 2, 3, 0, 2, 1, 2, 1, 3, 0, 0, 2, 1, 3, 1, 3, 0, 2, 3, 1, 2, 0,
0, 2, 1, 3, 2, 0, 3, 1, 1, 3, 0, 2, 3, 1, 2, 0, 0, 2, 3, 1, 2, 0, 1, 3, 1, 3, 0, 2, 3, 1, 2, 0, 2, 0,
1, 3, 0, 2, 3, 1, 1, 3, 0, 2, 3, 1, 2, 0, 2, 0, 3, 1, 0, 2, 1, 3, 1, 3, 0, 2, 3, 2, 1, 0, 0, 1, 2, 3,
2, 0, 3, 1, 1, 3, 0, 2, 3, 2, 1, 0, 2, 0, 3, 1, 0, 1, 2, 3, 1, 3, 2, 0, 0, 1, 3, 2, 2, 0, 1, 3, 3, 2,
0, 1, 1, 3, 2, 0, 0, 1, 3, 2, 3, 2, 0, 1, 2, 0, 1, 3, 1, 3, 2, 0, 0, 2, 1, 3, 2, 0, 3, 1, 3, 1, 0, 2,
1, 3, 2, 0, 0, 2, 1, 3, 3, 1, 0, 2, 2, 0, 3, 1, 1, 3, 2, 0, 0, 2, 3, 1, 2, 0, 1, 3, 3, 1, 0, 2, 1, 3,
2, 0, 0, 2, 3, 1, 2, 1, 0, 3, 3, 0, 1, 2, 1, 3, 2, 0, 0, 2, 3, 1, 3, 0, 1, 2, 2, 1, 0, 3, 1, 3, 2, 0,
0, 2, 3, 1, 3, 1, 0, 2, 2, 0, 1, 3, 1, 3, 2, 0, 2, 0, 1, 3, 0, 1, 3, 2, 3, 2, 0, 1, 1, 3, 2, 0, 2, 0,
1, 3, 0, 2, 3, 1, 3, 1, 0, 2, 1, 3, 2, 0, 2, 0, 1, 3, 3, 1, 0, 2, 0, 2, 3, 1, 2, 0, 1, 3, 3, 2, 0, 1,
3, 2, 0, 1, 0, 1, 3, 2, 1, 3, 2, 0, 2, 0, 3, 1, 0, 2, 1, 3, 3, 1, 0, 2, 1, 3, 2, 0, 2, 0, 3, 1, 3, 1,
0, 2, 0, 2, 1, 3, 1, 3, 2, 0, 2, 1, 0, 3, 0, 2, 3, 1, 3, 0, 1, 2, 1, 3, 2, 0, 2, 1, 0, 3, 3, 0, 1, 2,
0, 2, 3, 1, 1, 3, 2, 0, 3, 0, 1, 2, 0, 2, 3, 1, 2, 1, 0, 3, 1, 3, 2, 0, 3, 0, 1, 2, 2, 1, 0, 3, 0, 2,
3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 0, 2, 1, 3, 2, 0, 3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 0, 2, 3, 1, 2, 0, 1, 3,
1, 3, 2, 0, 3, 1, 0, 2, 2, 0, 1, 3, 0, 2, 3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 2, 0, 3, 1, 0, 2, 1, 3, 1, 3,
2, 0, 3, 2, 0, 1, 0, 1, 3, 2, 2, 0, 1, 3, 1, 3, 2, 0, 3, 2, 0, 1, 2, 0, 1, 3, 0, 1, 3, 2, 2, 0, 1, 3,
0, 1, 3, 2, 1, 3, 2, 0, 3, 2, 0, 1, 2, 0, 1, 3, 0, 1, 3, 2, 3, 2, 0, 1, 1, 3, 2, 0, 2, 0, 1, 3, 0, 2,
3, 1, 1, 3, 0, 2, 3, 1, 2, 0, 2, 0, 1, 3, 0, 2, 3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 0, 2, 3, 1, 2, 0, 1, 3,
3, 1, 0, 2, 1, 3, 2, 0, 2, 0, 1, 3, 0, 2, 3, 1, 3, 1, 2, 0, 1, 3, 0, 2, 2, 0, 1, 3, 0, 3, 2, 1, 1, 2,
3, 0, 3, 1, 0, 2, 2, 0, 1, 3, 0, 3, 2, 1, 3, 1, 0, 2, 1, 2, 3, 0, 2, 0, 1, 3, 1, 2, 3, 0, 0, 3, 2, 1,
3, 1, 0, 2, 2, 0, 1, 3, 1, 2, 3, 0, 3, 1, 0, 2, 0, 3, 2, 1, 2, 0, 1, 3, 1, 3, 2, 0, 2, 3, 1, 3, 1, 1,
2, 0, 2, 0, 1, 3, 1, 3, 0, 2, 3, 1, 2, 0, 0, 2, 3, 1, 2, 0, 1, 3, 1, 3, 2, 0, 0, 1, 3, 2, 3, 2, 0, 1,
2, 0, 1, 3, 1, 3, 2, 0, 0, 2, 3, 1, 3, 1, 0, 2, 2, 0, 1, 3, 1, 3, 2, 0, 3, 1, 0, 2, 0, 2, 3, 1, 2, 0,

1, 3, 1, 3, 2, 0, 3, 2, 0, 1, 0, 1, 3, 2, 2, 0, 1, 3, 3, 1, 0, 2, 0, 2, 3, 1, 1, 3, 2, 0, 2, 0, 1, 3,
3, 1, 0, 2, 0, 3, 2, 1, 1, 2, 3, 0, 2, 0, 1, 3, 3, 1, 0, 2, 1, 2, 3, 0, 0, 3, 2, 1, 2, 0, 1, 3, 3, 1,
0, 2, 1, 3, 2, 0, 0, 2, 3, 1, 2, 0, 1, 3, 3, 1, 2, 0, 0, 2, 3, 1, 1, 3, 0, 2, 2, 0, 1, 3, 3, 1, 2, 0,
1, 3, 0, 2, 0, 2, 3, 1, 2, 0, 1, 3, 3, 2, 0, 1, 0, 1, 3, 2, 1, 3, 2, 0, 2, 0, 1, 3, 3, 2, 0, 1, 1, 3,
2, 0, 0, 1, 3, 2, 2, 0, 3, 1, 0, 1, 2, 3, 1, 3, 0, 2, 3, 2, 1, 0, 2, 0, 3, 1, 0, 1, 2, 3, 3, 2, 1, 0,
1, 3, 0, 2, 2, 0, 3, 1, 0, 2, 1, 3, 1, 3, 0, 2, 3, 1, 2, 0, 2, 0, 3, 1, 0, 2, 1, 3, 1, 3, 2, 0, 3, 1,
0, 2, 2, 0, 3, 1, 0, 2, 1, 3, 3, 1, 0, 2, 1, 3, 2, 0, 2, 0, 3, 1, 0, 2, 1, 3, 3, 1, 2, 0, 1, 3, 0, 2,
2, 0, 3, 1, 0, 3, 1, 2, 1, 2, 0, 3, 3, 1, 2, 0, 2, 0, 3, 1, 0, 3, 1, 2, 3, 1, 2, 0, 1, 2, 0, 3, 2, 0,
3, 1, 1, 2, 0, 3, 0, 3, 1, 2, 3, 1, 2, 0, 2, 0, 3, 1, 1, 2, 0, 3, 3, 1, 2, 0, 0, 3, 1, 2, 2, 0, 3, 1,
1, 3, 0, 2, 0, 1, 2, 3, 3, 2, 1, 0, 2, 0, 3, 1, 1, 3, 0, 2, 0, 2, 1, 3, 3, 1, 2, 0, 2, 0, 3, 1, 1, 3,
0, 2, 3, 1, 2, 0, 0, 2, 1, 3, 2, 0, 3, 1, 1, 3, 0, 2, 3, 2, 1, 0, 0, 1, 2, 3, 2, 0, 3, 1, 1, 3, 2, 0,
0, 2, 1, 3, 3, 1, 0, 2, 2, 0, 3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 0, 2, 1, 3, 2, 0, 3, 1, 3, 1, 0, 2, 0, 2,
1, 3, 1, 3, 2, 0, 2, 0, 3, 1, 3, 1, 0, 2, 1, 3, 2, 0, 0, 2, 1, 3, 2, 0, 3, 1, 3, 1, 2, 0, 0, 2, 1, 3,
1, 3, 0, 2, 2, 0, 3, 1, 3, 1, 2, 0, 0, 3, 1, 2, 1, 2, 0, 3, 2, 0, 3, 1, 3, 1, 2, 0, 1, 2, 0, 3, 0, 3,
1, 2, 2, 0, 3, 1, 3, 1, 2, 0, 1, 3, 0, 2, 0, 2, 1, 3, 2, 0, 3, 1, 3, 2, 1, 0, 0, 1, 2, 3, 1, 3, 0, 2,
2, 0, 3, 1, 3, 2, 1, 0, 1, 3, 0, 2, 0, 1, 2, 3, 2, 1, 0, 3, 0, 2, 3, 1, 1, 3, 2, 0, 3, 0, 1, 2, 2, 1,
0, 3, 0, 2, 3, 1, 3, 0, 1, 2, 1, 3, 2, 0, 2, 1, 0, 3, 0, 3, 1, 2, 3, 0, 3, 0, 3, 2, 1, 2, 1, 0, 3, 0,
0, 3, 1, 2, 3, 0, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 0, 3, 2, 1, 1, 0, 3, 2, 3, 2, 1, 0, 2, 1, 0, 3, 0, 3,
2, 1, 1, 2, 3, 0, 3, 0, 1, 2, 2, 1, 0, 3, 0, 3, 2, 1, 3, 0, 1, 2, 1, 2, 3, 0, 2, 1, 0, 3, 0, 3, 2, 1,
3, 2, 1, 0, 1, 0, 3, 2, 2, 1, 0, 3, 1, 0, 3, 2, 0, 3, 2, 1, 3, 2, 1, 0, 2, 1, 0, 3, 1, 0, 3, 2, 3, 2,
1, 0, 0, 3, 2, 1, 2, 1, 0, 3, 1, 2, 3, 0, 0, 3, 1, 2, 3, 0, 2, 1, 2, 1, 0, 3, 1, 0, 3, 2, 3, 2, 1,
3, 0, 1, 2, 2, 1, 0, 3, 1, 2, 3, 0, 3, 0, 1, 2, 0, 3, 2, 1, 2, 1, 0, 3, 1, 2, 3, 0, 3, 0, 2, 1, 0, 3,
1, 2, 2, 1, 0, 3, 1, 3, 2, 0, 0, 2, 3, 1, 3, 0, 1, 2, 2, 1, 0, 3, 1, 3, 2, 0, 3, 0, 1, 2, 0, 2, 3, 1,
2, 1, 0, 3, 3, 0, 1, 2, 0, 2, 3, 1, 1, 3, 2, 0, 2, 1, 0, 3, 3, 0, 1, 2, 0, 3, 2, 1, 1, 2, 3, 0, 2, 1,
0, 3, 3, 0, 1, 2, 1, 2, 3, 0, 0, 3, 2, 1, 2, 1, 0, 3, 3, 0, 1, 2, 1, 3, 2, 0, 0, 2, 3, 1, 2, 1, 0, 3,
3, 0, 2, 1, 0, 3, 1, 2, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 2, 1, 1, 2, 3, 0, 0, 3, 1, 2, 2, 1, 0, 3, 3, 2,
1, 0, 0, 3, 2, 1, 1, 0, 3, 2, 2, 1, 0, 3, 3, 2, 1, 0, 1, 0, 3, 2, 0, 3, 2, 1, 2, 1, 3, 0, 0, 2, 1, 3,
1, 3, 0, 2, 3, 0, 2, 1, 2, 1, 3, 0, 0, 2, 1, 3, 3, 0, 2, 1, 1, 3, 0, 2, 2, 1, 3, 0, 0, 3, 1, 2, 1, 0,
2, 3, 3, 2, 0, 1, 2, 1, 3, 0, 0, 3, 1, 2, 1, 2, 0, 3, 3, 0, 2, 1, 2, 1, 3, 0, 0, 3, 1, 2, 3, 0, 2, 1,
1, 2, 0, 3, 2, 1, 3, 0, 0, 3, 1, 2, 3, 2, 0, 1, 1, 0, 2, 3, 2, 1, 3, 0, 0, 3, 2, 1, 1, 2, 0, 3, 3, 0,
1, 2, 2, 1, 3, 0, 0, 3, 2, 1, 3, 0, 1, 2, 1, 2, 0, 3, 2, 1, 3, 0, 1, 0, 2, 3, 0, 3, 1, 2, 3, 2, 0, 1,
2, 1, 3, 0, 1, 0, 2, 3, 3, 2, 0, 1, 0, 3, 1, 2, 2, 1, 3, 0, 1, 2, 0, 3, 0, 3, 1, 2, 3, 0, 2, 1, 2, 1,
3, 0, 1, 2, 0, 3, 0, 3, 2, 1, 3, 0, 1, 2, 2, 1, 3, 0, 1, 2, 0, 3, 3, 0, 1, 2, 0, 3, 2, 1, 2, 1, 3, 0,
1, 2, 0, 3, 3, 0, 2, 1, 0, 3, 1, 2, 3, 2, 0, 1, 1, 0, 2, 3, 2, 1, 3, 0, 0, 3, 2, 1, 1, 2, 0, 3, 3, 0,
0, 2, 3, 0, 2, 1, 0, 2, 1, 3, 2, 1, 3, 0, 3, 0, 1, 2, 0, 3, 2, 1, 1, 2, 0, 3, 2, 1, 3, 0, 3, 0, 1, 2,
1, 2, 0, 3, 0, 3, 2, 1, 2, 1, 3, 0, 3, 0, 2, 1, 0, 2, 1, 3, 1, 3, 0, 2, 2, 1, 3, 0, 3, 0, 2, 1, 0, 3,
1, 2, 1, 2, 0, 3, 2, 1, 3, 0, 3, 0, 2, 1, 1, 2, 0, 3, 0, 3, 1, 2, 2, 1, 3, 0, 3, 0, 2, 1, 1, 3, 0, 2,
0, 2, 1, 3, 2, 1, 3, 0, 3, 2, 0, 1, 0, 3, 1, 2, 1, 0, 2, 3, 2, 1, 3, 0, 3, 2, 0, 1, 1, 0, 2, 3, 0, 3,
1, 2, 2, 3, 0, 1, 0, 1, 2, 3, 1, 0, 3, 2, 3, 2, 1, 0, 2, 3, 0, 1, 0, 1, 2, 3, 1, 2, 3, 0, 3, 0, 1, 2,
2, 3, 0, 1, 0, 1, 2, 3, 3, 0, 1, 2, 1, 2, 3, 0, 2, 3, 0, 1, 0, 1, 2, 3, 3, 2, 1, 0, 1, 0, 3, 2, 2, 3,
0, 1, 0, 1, 3, 2, 1, 0, 2, 3, 3, 2, 1, 0, 2, 3, 0, 1, 0, 1, 3, 2, 3, 2, 1, 0, 1, 0, 2, 3, 2, 3, 0, 1,
0, 2, 1, 3, 1, 0, 3, 2, 3, 1, 2, 0, 2, 3, 0, 1, 0, 2, 1, 3, 3, 1, 2, 0, 1, 0, 3, 2, 2, 3, 0, 1, 1, 0,
2, 3, 0, 1, 3, 2, 3, 2, 1, 0, 2, 3, 0, 1, 1, 0, 2, 3, 3, 2, 1, 0, 0, 1, 3, 2, 2, 3, 0, 1, 1, 0, 3, 2,
0, 1, 2, 3, 3, 2, 1, 0, 2, 3, 0, 1, 1, 0, 3, 2, 0, 2, 1, 3, 3, 1, 2, 0, 2, 3, 0, 1, 1, 0, 3, 2, 3, 1,
2, 0, 0, 2, 1, 3, 2, 3, 0, 1, 1, 0, 3, 2, 3, 2, 1, 0, 0, 1, 2, 3, 2, 3, 0, 1, 1, 2, 3, 0, 0, 1, 2, 3,
3, 0, 1, 2, 2, 3, 0, 1, 1, 2, 3, 0, 3, 0, 1, 2, 0, 1, 2, 3, 2, 3, 0, 1, 3, 0, 1, 2, 0, 1, 2, 3, 1, 2,
3, 0, 2, 3, 0, 1, 3, 0, 1, 2, 1, 2, 3, 0, 0, 1, 2, 3, 2, 3, 0, 1, 3, 1, 2, 0, 0, 2, 1, 3, 1, 0, 3, 2,
2, 3, 0, 1, 3, 1, 2, 0, 1, 0, 3, 2, 0, 2, 1, 3, 2, 3, 0, 1, 3, 2, 1, 0, 0, 1, 2, 3, 1, 0, 3, 2, 2, 3,
0, 1, 3, 2, 1, 0, 0, 1, 3, 2, 1, 0, 2, 3, 2, 3, 0, 1, 3, 2, 1, 0, 1, 0, 2, 3, 0, 1, 3, 2, 2, 3, 0, 1,
3, 2, 1, 0, 1, 0, 3, 2, 0, 1, 2, 3, 2, 3, 1, 0, 0, 1, 2, 3, 1, 0, 3, 2, 3, 2, 0, 1, 2, 3, 1, 0, 0, 1,
2, 3, 3, 2, 0, 1, 1, 0, 3, 2, 2, 3, 1, 0, 0, 1, 3, 2, 1, 0, 2, 3, 3, 2, 0, 1, 2, 3, 1, 0, 0, 1, 3, 2,
1, 2, 0, 3, 3, 0, 2, 1, 2, 3, 1, 0, 0, 1, 3, 2, 3, 0, 2, 1, 1, 2, 0, 3, 2, 3, 1, 0, 2, 1, 3, 2, 3, 2,
0, 1, 1, 0, 2, 3, 2, 3, 1, 0, 0, 2, 3, 1, 1, 0, 2, 3, 3, 1, 0, 2, 2, 3, 1, 0, 0, 2, 3, 1, 3, 1, 0, 2,
1, 0, 2, 3, 2, 3, 1, 0, 1, 0, 2, 3, 0, 1, 3, 2, 3, 2, 0, 1, 2, 3, 1, 0, 1, 0, 2, 3, 0, 2, 3, 1, 3, 1,
0, 2, 2, 3, 1, 0, 1, 0, 2, 3, 3, 1, 0, 2, 0, 2, 3, 1, 2, 3, 1, 0, 1, 0, 2, 3, 3, 2, 0, 1, 0, 1, 3, 2,
2, 3, 1, 0, 1, 0, 3, 2, 0, 1, 2, 3, 3, 2, 0, 1, 2, 3, 1, 0, 1, 0, 3, 2, 3, 2, 0, 1, 0, 1, 2, 3, 2, 3,
1, 0, 1, 2, 0, 3, 0, 1, 3, 2, 3, 0, 2, 1, 2, 3, 1, 0, 1, 2, 0, 3, 3, 0, 2, 1, 0, 1, 3, 2, 2, 3, 1, 0,
3, 0, 2, 1, 0, 1, 3, 2, 1, 2, 0, 3, 2, 3, 1, 0, 3, 0, 2, 1, 1, 2, 0, 3, 0, 1, 3, 2, 2, 3, 1, 0, 3, 1,
0, 2, 0, 2, 3, 1, 1, 0, 2, 3, 2, 3, 1, 0, 3, 1, 0, 2, 1, 0, 2, 3, 0, 2, 3, 1, 2, 3, 1, 0, 3, 2, 0, 1,
0, 1, 2, 3, 1, 0, 3, 2, 2, 3, 1, 0, 3, 2, 0, 1, 0, 1, 3, 2, 1, 0, 2, 3, 2, 3, 1, 0, 3, 2, 0, 1, 1, 0,
2, 3, 0, 1, 3, 2, 2, 3, 1, 0, 3, 2, 0, 1, 1, 0, 3, 2, 0, 1, 2, 3, 3, 0, 1, 2, 0, 1, 2, 3, 1, 2, 3, 0,
2, 3, 0, 1, 3, 0, 1, 2, 0, 1, 2, 3, 2, 3, 0, 1, 1, 2, 3, 0, 3, 0, 1, 2, 0, 2, 3, 1, 1, 3, 2, 0, 2, 1,
0, 3, 3, 0, 1, 2, 0, 2, 3, 1, 2, 1, 0, 3, 1, 3, 2, 0, 3, 0, 2, 3, 1, 2, 1, 0, 3, 2, 1, 1, 2, 0, 3, 0,
3, 0, 1, 2, 0, 3, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 1, 2, 0, 3, 2, 1, 2, 1, 0, 3, 1, 2, 3, 0, 3, 0,
1, 2, 0, 3, 2, 1, 2, 1, 3, 0, 1, 2, 0, 3, 3, 0, 1, 2, 1, 2, 0, 3, 0, 3, 2, 1, 2, 1, 3, 0, 3, 0, 1, 2,
1, 2, 0, 3, 2, 1, 3, 0, 0, 3, 2, 1, 3, 0, 1, 2, 1, 2, 3, 0, 0, 1, 2, 3, 2, 3, 0, 1, 2, 3, 2, 1, 3, 0,
3, 0, 0, 3, 2, 1, 2, 1, 0, 3, 3, 0, 1, 2, 1, 2, 3, 0, 2, 1, 0, 3, 0, 3, 2, 1, 3, 0, 1, 2, 1, 2, 3, 0,
2, 3, 0, 1, 0, 1, 2, 3, 3, 0, 1, 2, 1, 3, 2, 0, 0, 2, 3, 1, 2, 1, 0, 3, 3, 0, 1, 2, 1, 3, 2, 0, 2, 1,
0, 3, 0, 2, 3, 1, 3, 0, 1, 2, 2, 1, 0, 3, 0, 2, 3, 1, 1, 3, 2, 0, 3, 0, 1, 2, 2, 1, 0, 3, 0, 3, 2, 1,

```
1, 2, 3, 0, 3, 0, 1, 2, 2, 1, 0, 3, 1, 2, 3, 0, 0, 3, 2, 1, 3, 0, 1, 2, 2, 1, 0, 3, 1, 3, 2, 0, 0, 2,
3, 1, 3, 0, 1, 2, 2, 1, 3, 0, 0, 3, 2, 1, 1, 2, 0, 3, 3, 0, 1, 2, 2, 1, 3, 0, 1, 2, 0, 3, 0, 3, 2, 1,
3, 0, 1, 2, 2, 3, 0, 1, 0, 1, 2, 3, 1, 2, 3, 0, 3, 0, 1, 2, 2, 3, 0, 1, 1, 2, 3, 0, 0, 1, 2, 3, 3, 0,
2, 1, 0, 1, 3, 2, 1, 2, 0, 3, 2, 3, 1, 0, 3, 0, 2, 1, 0, 1, 3, 2, 2, 3, 1, 0, 1, 2, 0, 3, 3, 0, 2, 1,
0, 2, 1, 3, 1, 3, 0, 2, 2, 1, 3, 0, 3, 0, 2, 1, 0, 2, 1, 3, 2, 1, 3, 0, 1, 3, 0, 2, 3, 0, 2, 1, 0, 3,
1, 2, 1, 2, 0, 3, 2, 1, 3, 0, 3, 0, 2, 1, 0, 3, 1, 2, 1, 2, 3, 0, 2, 1, 0, 3, 3, 0, 2, 1, 0, 3, 1, 2,
2, 1, 0, 3, 1, 2, 3, 0, 3, 0, 2, 1, 0, 3, 1, 2, 2, 1, 3, 0, 1, 2, 0, 3, 3, 0, 2, 1, 1, 2, 0, 3, 0, 1,
3, 2, 2, 3, 1, 0, 3, 0, 2, 1, 1, 2, 0, 3, 0, 3, 1, 2, 2, 1, 3, 0, 3, 0, 2, 1, 1, 2, 0, 3, 2, 1, 3, 0,
0, 3, 1, 2, 3, 0, 2, 1, 1, 2, 0, 3, 2, 3, 1, 0, 0, 1, 3, 2, 3, 0, 2, 1, 1, 2, 3, 0, 0, 3, 1, 2, 2, 1,
0, 3, 3, 0, 2, 1, 1, 2, 3, 0, 2, 1, 0, 3, 0, 3, 1, 2, 3, 0, 2, 1, 1, 3, 0, 2, 0, 2, 1, 3, 2, 1, 3, 0,
3, 0, 2, 1, 1, 3, 0, 2, 2, 1, 3, 0, 0, 2, 1, 3, 3, 0, 2, 1, 2, 1, 0, 3, 0, 3, 1, 2, 1, 2, 3, 0, 3, 0,
2, 1, 2, 1, 0, 3, 1, 2, 3, 0, 0, 3, 1, 2, 3, 0, 2, 1, 2, 1, 3, 0, 0, 2, 1, 3, 1, 3, 0, 2, 3, 0, 2, 1,
2, 1, 3, 0, 0, 3, 1, 2, 1, 2, 0, 3, 3, 0, 2, 1, 2, 1, 3, 0, 1, 2, 0, 3, 0, 3, 1, 2, 3, 0, 2, 1, 2, 1,
3, 0, 1, 3, 0, 2, 0, 2, 1, 3, 3, 0, 2, 1, 2, 3, 1, 0, 0, 1, 3, 2, 1, 2, 0, 3, 3, 0, 2, 1, 2, 3, 1, 0,
1, 2, 0, 3, 0, 1, 3, 2, 3, 1, 0, 2, 0, 2, 1, 3, 1, 3, 2, 0, 2, 0, 3, 1, 3, 1, 0, 2, 0, 2, 1, 3, 2, 0,
3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 0, 2, 3, 1, 1, 0, 2, 3, 2, 3, 1, 0, 3, 1, 0, 2, 0, 2, 3, 1, 1, 3, 2, 0,
2, 0, 1, 3, 3, 1, 0, 2, 0, 2, 3, 1, 2, 0, 1, 3, 1, 3, 2, 0, 3, 1, 0, 2, 0, 2, 3, 1, 2, 0, 1, 0, 2, 0,
2, 3, 3, 1, 0, 2, 0, 3, 2, 1, 1, 2, 3, 0, 2, 0, 1, 3, 3, 1, 0, 2, 0, 3, 2, 1, 2, 0, 1, 3, 1, 2, 3, 0,
3, 1, 0, 2, 1, 0, 2, 3, 0, 2, 3, 1, 2, 3, 1, 0, 3, 1, 0, 2, 1, 0, 2, 3, 2, 3, 1, 0, 0, 2, 3, 1, 3, 1,
0, 2, 1, 2, 3, 0, 0, 3, 2, 1, 2, 0, 1, 3, 3, 1, 0, 2, 1, 2, 3, 0, 2, 0, 1, 3, 0, 3, 2, 1, 3, 1, 0, 2,
1, 3, 2, 0, 0, 2, 1, 3, 2, 0, 3, 1, 3, 1, 0, 2, 1, 3, 2, 0, 0, 2, 3, 1, 2, 0, 1, 3, 3, 1, 0, 2, 1, 3,
2, 0, 2, 0, 1, 3, 0, 2, 3, 1, 3, 1, 0, 2, 1, 3, 2, 0, 2, 0, 3, 1, 0, 2, 1, 3, 3, 1, 0, 2, 2, 0, 1, 3,
0, 2, 3, 1, 1, 3, 2, 0, 3, 1, 0, 2, 2, 0, 1, 3, 0, 3, 2, 1, 1, 2, 3, 0, 3, 1, 0, 2, 2, 0, 1, 3, 1, 2,
3, 0, 0, 3, 2, 1, 3, 1, 0, 2, 2, 0, 1, 3, 1, 3, 2, 0, 0, 2, 3, 1, 3, 1, 0, 2, 2, 0, 3, 1, 0, 2, 1, 3,
1, 3, 2, 0, 3, 1, 0, 2, 2, 0, 3, 1, 1, 3, 2, 0, 0, 2, 1, 3, 3, 1, 0, 2, 2, 3, 1, 0, 0, 2, 3, 1, 1, 0,
2, 3, 3, 1, 0, 2, 2, 3, 1, 0, 1, 0, 2, 3, 0, 2, 3, 1, 3, 1, 2, 0, 0, 2, 1, 3, 1, 0, 3, 2, 2, 3, 0, 1,
3, 1, 2, 0, 0, 2, 1, 3, 1, 3, 0, 2, 2, 0, 3, 1, 3, 1, 2, 0, 0, 2, 1, 3, 2, 0, 3, 1, 1, 3, 0, 2, 3, 1,
2, 0, 0, 2, 1, 3, 2, 3, 0, 1, 1, 0, 3, 2, 3, 1, 2, 0, 0, 2, 3, 1, 1, 3, 0, 2, 2, 0, 1, 3, 3, 1, 2, 0,
0, 2, 3, 1, 2, 0, 1, 3, 1, 3, 0, 2, 3, 1, 2, 0, 0, 3, 1, 2, 1, 2, 0, 3, 2, 0, 3, 1, 3, 1, 2, 0, 0, 3,
1, 2, 2, 0, 3, 1, 1, 2, 0, 3, 3, 1, 2, 0, 1, 0, 3, 2, 0, 2, 1, 3, 2, 3, 0, 1, 3, 1, 2, 0, 1, 0, 3, 2,
2, 3, 0, 1, 0, 2, 1, 3, 3, 1, 2, 0, 1, 2, 0, 3, 0, 3, 1, 2, 2, 0, 3, 1, 3, 1, 2, 0, 1, 2, 0, 3, 2, 0,
3, 1, 0, 3, 1, 2, 3, 1, 2, 0, 1, 3, 0, 2, 0, 2, 1, 3, 2, 0, 3, 1, 3, 1, 2, 0, 1, 3, 0, 2, 0, 2, 3, 1,
2, 0, 1, 3, 3, 1, 2, 0, 1, 3, 0, 2, 2, 0, 1, 3, 0, 2, 3, 1, 3, 1, 2, 0, 1, 3, 0, 2, 2, 0, 3, 1, 0, 2,
1, 3, 3, 1, 2, 0, 2, 0, 1, 3, 0, 2, 3, 1, 1, 3, 0, 2, 3, 1, 2, 0, 2, 0, 1, 3, 1, 3, 0, 2, 0, 2, 3, 1,
3, 1, 2, 0, 2, 0, 3, 1, 0, 2, 1, 3, 1, 3, 0, 2, 3, 1, 2, 0, 2, 0, 3, 1, 0, 3, 1, 2, 1, 2, 0, 3, 3, 1,
2, 0, 2, 0, 3, 1, 1, 2, 0, 3, 0, 3, 1, 2, 3, 1, 2, 0, 2, 0, 3, 1, 1, 3, 0, 2, 0, 2, 1, 3, 3, 1, 2, 0,
2, 3, 0, 1, 0, 2, 1, 3, 1, 0, 3, 2, 3, 1, 2, 0, 2, 3, 0, 1, 1, 0, 3, 2, 0, 2, 1, 3, 3, 2, 0, 1, 0, 1,
2, 3, 1, 0, 3, 2, 2, 3, 1, 0, 3, 2, 0, 1, 0, 1, 2, 3, 2, 3, 1, 0, 1, 0, 3, 2, 3, 2, 0, 1, 0, 1, 3, 2,
1, 0, 2, 3, 2, 3, 1, 0, 3, 2, 0, 1, 0, 1, 3, 2, 1, 3, 2, 0, 2, 0, 1, 3, 3, 2, 0, 1, 0, 1, 3, 2, 2, 0,
1, 3, 1, 3, 2, 0, 3, 2, 0, 1, 0, 1, 3, 2, 2, 3, 1, 0, 1, 0, 2, 3, 3, 2, 0, 1, 0, 3, 1, 2, 1, 0, 2, 3,
2, 1, 3, 0, 3, 2, 0, 1, 0, 3, 1, 2, 2, 1, 3, 0, 1, 0, 2, 3, 3, 2, 0, 1, 1, 0, 2, 3, 0, 1, 3, 2, 2, 3,
1, 0, 3, 2, 0, 1, 1, 0, 2, 3, 0, 3, 1, 2, 2, 1, 3, 0, 3, 2, 0, 1, 1, 0, 2, 3, 2, 1, 3, 0, 0, 3, 1, 2,
3, 2, 0, 1, 1, 0, 2, 3, 2, 3, 1, 0, 0, 1, 3, 2, 3, 2, 0, 1, 1, 0, 3, 2, 0, 1, 2, 3, 2, 3, 1, 0, 3, 2,
0, 1, 1, 0, 3, 2, 2, 3, 1, 0, 0, 1, 2, 3, 3, 2, 0, 1, 1, 3, 2, 0, 0, 1, 3, 2, 2, 0, 1, 3, 3, 2, 0, 1,
1, 3, 2, 0, 2, 0, 1, 3, 0, 1, 3, 2, 3, 2, 0, 1, 2, 0, 1, 3, 0, 1, 3, 2, 1, 3, 2, 0, 3, 2, 0, 1, 2, 0,
1, 3, 1, 3, 2, 0, 0, 1, 3, 2, 3, 2, 0, 1, 2, 1, 3, 0, 0, 3, 1, 2, 1, 0, 2, 3, 3, 2, 0, 1, 2, 1, 3, 0,
1, 0, 2, 3, 0, 3, 1, 2, 3, 2, 0, 1, 2, 3, 1, 0, 0, 1, 2, 3, 1, 0, 3, 2, 3, 2, 0, 1, 2, 3, 1, 0, 0, 1,
3, 2, 1, 0, 2, 3, 3, 2, 0, 1, 2, 3, 1, 0, 1, 0, 2, 3, 0, 1, 3, 2, 3, 2, 0, 1, 2, 3, 1, 0, 1, 0, 3, 2,
0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 1, 0, 3, 2, 2, 3, 0, 1, 3, 2, 1, 0, 0, 1, 2, 3, 1, 3, 0, 2, 2, 0,
3, 1, 3, 2, 1, 0, 0, 1, 2, 3, 2, 0, 3, 1, 1, 3, 0, 2, 3, 2, 1, 0, 0, 1, 2, 3, 1, 1, 0, 3, 2, 3, 2, 0,
3, 2, 1, 0, 0, 1, 3, 2, 1, 0, 2, 3, 2, 3, 0, 1, 3, 2, 1, 0, 0, 1, 3, 2, 2, 3, 0, 1, 1, 0, 2, 3, 3, 2,
1, 0, 0, 3, 2, 1, 1, 0, 3, 2, 2, 1, 0, 3, 3, 2, 1, 0, 0, 3, 2, 1, 2, 1, 0, 3, 1, 0, 3, 2, 3, 2, 1, 0,
1, 0, 2, 3, 0, 1, 3, 2, 2, 3, 0, 1, 3, 2, 1, 0, 1, 0, 2, 3, 2, 3, 0, 1, 0, 1, 3, 2, 3, 2, 1, 0, 1, 0,
3, 2, 0, 1, 2, 3, 2, 3, 0, 1, 3, 2, 1, 0, 1, 0, 3, 2, 0, 3, 2, 1, 2, 1, 0, 3, 3, 2, 1, 0, 1, 0, 3, 2,
2, 1, 0, 3, 0, 3, 2, 1, 3, 2, 1, 0, 1, 0, 3, 2, 2, 3, 0, 1, 0, 1, 2, 3, 3, 2, 1, 0, 2, 0, 1, 3, 0, 2,
2, 3, 2, 0, 3, 1, 3, 2, 1, 0, 1, 3, 0, 2, 2, 0, 3, 1, 0, 1, 2, 3, 3, 2, 1, 0, 2, 0, 3, 1, 0, 1, 2, 3,
1, 3, 0, 2, 3, 2, 1, 0, 2, 0, 3, 1, 1, 3, 0, 2, 0, 1, 2, 3, 3, 2, 1, 0, 2, 1, 0, 3, 0, 3, 2, 1, 1, 0,
3, 2, 3, 2, 1, 0, 2, 1, 0, 3, 1, 0, 3, 2, 0, 3, 2, 1, 3, 2, 1, 0, 2, 3, 0, 1, 0, 1, 2, 3, 1, 0, 3, 2,
3, 2, 1, 0, 2, 3, 0, 1, 0, 1, 3, 2, 1, 0, 2, 3, 3, 2, 1, 0, 2, 3, 0, 1, 1, 0, 2, 3, 0, 1, 3, 2, 3, 2,
1, 0, 2, 3, 0, 1, 1, 0, 3, 2, 0, 1, 2, 3 };
   double error = 0;
   double rate1 = 0, rate2 = 0, rate3 = 0, rate4 = 0, rate5 = 0, rate6 = 0;
   int decode1 = 0, decode2 = 0, decode3 = 0, decode4 = 0, decode5 = 0, decode6 = 0;
   int calculation = 100;
   int input[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
   int output[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
   unsigned long long init[4] = { 0x12345ULL, 0x23456ULL, 0x34567ULL, 0x45678ULL }, length = 4;
   init_by_array64(init, length);
   for (int i = 0; i < 100; i++) {
```

```
        error = (double)i / calculation;
        if (3 * error > 1) {
            break;
        }
        rate1 = 0, rate2 = 0, rate3 = 0, rate4 = 0, rate5 = 0, rate6 = 0;
        decode1 = 0, decode2 = 0, decode3 = 0, decode4 = 0, decode5 = 0, decode6 = 0;
        for (int i = 0; i < 576; i++) {
            for (int j = 0; j < 16; j++) {
                input[j] = latinsquare[i][j];
            }
            for (int j = 0; j < calculation; j++) {
                channel(input, output, error);
                if (decoding1(input, output, error, latinsquare) != 0) {
                    decode1++;
                }
                if (decoding2(input, output, error) != 0) {
                    decode2++;
                }
                if (decoding3(input, output, error) != 0) {
                    decode3++;
                }
                if (decoding4(input, output, error) != 0) {
                    decode4++;
                }
                if (decoding5(input, output, error) != 0) {
                    decode5++;
                }
                if (decoding6(input, output, error) != 0) {
                    decode6++;
                }
            }
        }
        rate1 = (double)decode1 / (calculation * 576);
        rate2 = (double)decode2 / (calculation * 576);
        rate3 = (double)decode3 / (calculation * 576);
        rate4 = (double)decode4 / (calculation * 576);
        rate5 = (double)decode5 / (calculation * 576);
        rate6 = (double)decode6 / (calculation * 576);
        printf("%f %f %f %f %f %f\n", error, rate1, rate2, rate4, rate5, rate6);
    }
}
```

A.3 sum-product の計算値を確認するプログラム

```
#include <stdio.h>
#pragma warning(disable : 4996)

int main() {
    int In[16] = { 0,1,2,3,1,3,0,2,2,3,0,1,3,2,1,0 };
    double Mxf[32][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    double Mfx[32][4] = { {1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},
{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1} };
    double Mx[16][4] = { {0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},
{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0} };
    double Chan[4][4] = { {0.7,0.1,0.1,0.1},{0.1,0.7,0.1,0.1},{0.1,0.1,0.7,0.1},{0.1,0.1,0.1,0.7} };
    int Mfxf[32] = { 16,20,24,28,17,21,25,29,18,22,26,30,19,23,27,31,0,4,8,12,1,5,9,13,2,6,
10,14,3,7,11,15 };
    int Mxfx[32][3] = { {1,2,3},{0,2,3},{0,1,3},{0,1,2},{5,6,7},{4,6,7},{4,5,7},{4,5,6},{9,10,11},
{8,10,11},{8,9,11},{8,9,10},{13,14,15},{12,14,15},{12,13,15},{12,13,14},{17,18,19},{16,18,19},
{16,17,19},{16,17,18},{21,22,23,},{20,22,23},{20,21,23},{20,21,22},{25,26,27},{24,26,27},
```

```
{24,25,27},{24,25,26},{29,30,31},{28,30,31},{28,29,31},{28,29,30} };
    int Wx[32] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15 };
    int Perm[6][3] = { {1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2,1} };
    int Mmar[16][2] = { {0,16},{1,20},{2,24},{3,28},{4,17},{5,21},{6,25},{7,29},{8,18},{9,22},
{10,26},{11,30},{12,19},{13,23},{14,27},{15,31} };
    int Deco[8][4] = { {0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15},{0,4,8,12},{1,5,9,13},
{2,6,10,14},{3,7,11,15} };
    int Calc = 0, Outp = 0;
    int Pmax[16] = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
loop:for (int i = 0; i < 32; i++) {
    for (int j = 0; j < 4; j++) {
        Mxf[i][j] = Chan[j][In[Wx[i]]] * Mfx[Mfxf[i]][j];
    }
}
for (int i = 0; i < 32; i++) {
    for (int j = 0; j < 4; j++) {
        double Sum = 0;
        for (int l = 0; l < 6; l++) {
            double Prod = 1;
            Prod *= Mxf[Mxfx[i][0]][(Perm[l][0] + j) % 4] *
 Mxf[Mxfx[i][1]][(Perm[l][1] + j) % 4] * Mxf[Mxfx[i][2]][(Perm[l][2] + j) % 4];
            Sum += Prod;
        }
        Mfx[i][j] = Sum;
    }
}
for (int i = 0; i < 16; i++) {
    for (int j = 0; j < 4; j++) {
        Mx[i][j] = Chan[j][In[Wx[i]]] * Mfx[Mmar[i][0]][j] * Mfx[Mmar[i][1]][j];
    }
}
for (int i = 0; i < 16; i++) {
    for (int j = 0; j < 4; j++) {
        if (Mx[i][j] > Mx[i][Pmax[i]]) {
            Pmax[i] = j;
        }
    }
}
for (int i = 0; i < 8; i++) {
    int x = 0, y = 0, z = 0, w = 0;
    for (int j = 0; j < 4; j++) {
        if (Pmax[Deco[i][j]] == 0) {
            x++;
        }
        else if (Pmax[Deco[i][j]] == 1) {
            y++;
        }
        else if (Pmax[Deco[i][j]] == 2) {
            z++;
        }
        else {
            w++;
        }
    }
    if (x != 1 || y != 1 || z != 1 || w != 1) {
        Calc++;
        if (Calc < 10) {
            printf("%g \n",Mx[0][0]);
            goto loop;
        }
        else {
            goto end;
        }
    }
}
end:printf("○から■ \n");
for (int i = 0; i < 32; i++) {
    printf("%d ", i);
    for (int j = 0; j < 4; j++) {
```

```
        printf("%g ", Mxf[i][j]);
    }
    printf("\n");
}
printf("■から○ \n");
for (int i = 0; i < 32; i++) {
    printf("%d ", i);
    for (int j = 0; j < 4; j++) {
        printf("%g ", Mfx[i][j]);
    }
    printf("\n");
}
printf("確率 \n");
for (int i = 0; i < 16; i++) {
    printf("%d ", i);
    for (int j = 0; j < 4; j++) {
        printf("%g ", Mx[i][j]);
    }
    printf("\n");
}
printf("ラテン方陣 \n");
for (int i = 0; i < 16; i++) {
    printf("%d ", Pmax[i]);
    Outp++;
    if (Outp > 3) {
        printf("\n");
        Outp = 0;
    }
}
printf("計算回数 %d\n", Calc);
}
```