信州大学工学部

学士論文

クラスカル法を用いて作成した
検査行列の性能評価に向けて

指導教員　西新 幹彦 准教授

学科　電子情報システム工学科
学籍番号　20T2094K
氏名　田村 涼真

2024 年 2 月 21 日

目次

1　はじめに

現代社会において，インターネットや電波を介した情報のやりとりが盛んに行われている．その際，送信者から受信者に情報を正しく送るために誤り訂正技術が導入されている．情報を伝送する際，符号器では誤り訂正のための冗長性が符号語に追加され，復号器では誤りを検出し訂正するためにその冗長性が利用される．誤り訂正技術は，我々の生活において必要不可欠であり，幅広く利用されている．通信を行う上で代表的な情報の符号化方法と復号方法として，LDPC 符号と sum-product 復号法がある．これらはそれぞれ生成行列，検査行列といった行列を用いる．検査行列は，2 部グラフで表現することができる．2 部グラフとは，頂点が 2 つの独立した集合に分かれ，異なる集合の頂点同士の間に限って辺が存在するグラフである．検査行列を 2 部グラフに変換したとき，長さの短いループが存在すると，sum-product 復号法により得られる復号結果に悪影響を及ぼすことが知られている [1].

本研究では，クラスカル法というアルゴリズムを用いることで，2 部グラフに変換したときループの存在しない検査行列を作成した．クラスカル法とは，グラフ理論において，重みつき連結グラフの最小全域木を求めるアルゴリズムの 1 つである．クラスカル法を用いることでループを形成するグラフの辺が剪定されていく．よって基となる初めのグラフを完全 2 部グラフとし，クラスカル法を用いて辺を剪定すると，ループのない検査行列を作成することができる．本研究では，2×3 の完全 2 部グラフから作成可能な行列のサイズが 2×3 のループのない検査行列を全て作成した．そして，作成した検査行列の復号性能を調べた．

2　誤り訂正符号の復号方法の原理

本章では，本研究で用いた通信路モデルと線形符号の原理について述べる．本章では，文献 [1], 文献 [2], 文献 [3], 文献 [4] に基づき，通信路や復号方法の原理について述べる．

2.1　通信路

本研究で想定した通信路は，図 1 のような 2 元対称通信路である．2 元対称通信路は，通信路として最もシンプルな通信路である．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通信路である．通信路モデルは，入力信号を表す確率変数 X と出力信号を表す確率変数 Y との間の条件付き確率を利用して記述される．つまり通信路の条件付き確率は，通信路が与えられた条件での信号の伝送確率を示す．

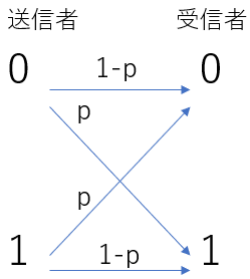

図 1　2 元対称通信路

2.2　線形符号

本研究は，線形符号の構成に関する研究である．本節では，文献 [2] に基づいて線形符号の概要を説明する．2 元ベクトル空間 $\mathbb{F}_2$ のある部分空間を C と表記する．これを $\mathbb{F}_2$ 上の線形符号と呼ぶ．符号 C の次元を k とするとき，符号 C はベクトル空間であるから，k 個の基底ベクトル $(\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_k)$ により張られる．2 元 $k \times n$ 行列 G を

$$G \triangleq \begin{pmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \vdots \\ \boldsymbol{g}_k \end{pmatrix} \tag{1}$$

と定義する．この行列 G を C の生成行列と呼ぶ．この定義より，符号 C の任意の符号語 c は

$$c = m_1\boldsymbol{g}_1 + m_2\boldsymbol{g}_2 + \cdots + m_k\boldsymbol{g}_k \tag{2}$$

のように基底ベクトルの線形結合として表せる．よって情報ベクトル $\boldsymbol{m} \triangleq (m_1, m_2, \ldots, m_k)$ を符号化したい場合，ベクトル $\boldsymbol{m}$ と生成行列 G の積

$$c = \boldsymbol{m}G \tag{3}$$

を計算すれば良い．このように生成行列が定まると情報ベクトルと符号語間に 1 対 1 の写像が定まる．次に，$m = n - k$ とする．2 元 $m \times n$ 行列 H がランク m を持ち，

$$GH^T = 0 \tag{4}$$

（H^T は行列 H の転置行列を表す）を満たすとき，H を符号 C の検査行列と呼ぶ．同じ符号 C を定義する生成行列 G と検査行列 H に成り立つ関係を考える．ある符号の $\mathbb{F}_2$ 上の生成行列が次の形を持つと仮定する．

$$G = (I_k \ P) \tag{5}$$

ここで I_k は $k \times k$ 単位行列であり，P は $k \times (n-k)$ 行列である．また，行列 P の ij 要素を $p_{i_j} \in \mathbb{F}_2$ とする．また，次の形の検査行列を考える．

$$H = (-P^t \ I_{n-k}) \tag{6}$$

ここで生成行列 G の i 行目の行ベクトル g_i と検査行列 H の j 行目の行ベクトル h_j の積に注目する．このとき，$g_i h_j^t = -p_{ij} + p_{ij} = 0$ が任意の i において成り立つ．

G により生成される符号の符号語を $\sum_{i=1}^{k} c_i g_i \ (c_i \in \mathbb{F}_2)$ としたとき，

$$H\left(\sum_{i=1}^{k} c_i g_i\right)^t = H\left(\sum_{i=1}^{k} c_i {g_i}^t\right) \tag{7}$$

$$= \left(\sum_{i=1}^{k} c_i h_1 {g_i}^t, \sum_{i=1}^{k} c_i h_2 {g_i}^t, \ldots, \sum_{i=1}^{k} c_i h_{n-k} {g_i}^t\right) \tag{8}$$

$$= 0 \tag{9}$$

が成り立つ．上式の変形では，関係 $g_i {h_j}^t = 0$ を利用している．すなわち，G の定義する線形符号 C_G の任意の符号語 c について，$Hc^t = 0$ が成り立つ．H の定義する線形符号を C_H とすると，以上より $C_G \subset C_H$ がいえる．次にそれぞれの符号の含む符号語数を考える．C_G については 2^k である．H のランクは $n-k$ であり，C_H の符号語数も 2^k である．したがって，$C_G = C_H$ とわかる．すなわち，生成行列 G と検査行列 H は同じ線形行列を定義している．また上式の関係式は，生成行列から検査行列またはその逆を求める手法を与えている．

本研究では，検査行列を作成し，それを元に生成行列を作成した．$\mathbb{F}_2$ 上で定義され，符号語長が 3 となるような線形符号を構成した．

2.3 LDPC 符号

LDPC 符号は，疎な検査行列により定義される線形符号である．文献 [1] に基づいて説明する．疎な行列とは，行列内の非零要素の数が非常に少ない行列を指す．本研究では，$\mathbb{F}_2$ 上の 2 元 LDPC 符号のみを扱うため，以下のように定義される．2 元 $m \times n$ 行列 $H(0 < m < n)$ が与えられたとき，この行列を検査行列とみなすことにより，2 元線形符号 C が

$$C \triangleq \{x \in \mathbb{F}_2^n : Hx^t = 0\} \tag{10}$$

と定義される．H が疎な行列の場合，C は LDPC 符号と呼ばれる．疎な行列を検査行列として用いる理由は，LDPC 符号の復号特性に関係する．LDPC 符号の復号には，sum-product アルゴリズムが利用される．本研究で作成した検査行列は 2 部グラフに変換したときループがない．また，2 部グラフに変換したとき全域木となっているため，行列内の非零要素の数が非

常に少ない．そのため，本研究で作成した検査行列を疎な行列とみなして LDPC 符号を定義した．

2.4 sum-product 復号法

LDPC 符号に対する代表的な復号法として sum-product 復号法がある．本節では，本研究で用いた確率領域 sum-product 復号法について文献 [1], 文献 [3] に基づいて説明する．

2 元 m 行 n 列の行列 H を復号したい LDPC 符号の検査行列とする．また，検査行列 H の m 行 n 列目要素 h_{mn} と表記する．整数の集合 $\{1, 2, \ldots, n\}$ の部分集合 $A(m)$, $B(n)$ を次のように定義する．

$$A(m) \triangleq \{n : h_{mn} = 1\} \tag{11}$$

$$B(n) \triangleq \{m : h_{mn} = 1\} \tag{12}$$

すなわち，$A(m)$ は検査行列 H の m 行目において，1 が立っている列インデックスの集合を意味し，$B(n)$ は検査行列 H の n 列目において 1 が立っている行インデックスの集合を指す．

受信語 $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ を受信したものと仮定する．また，2 元対称通信路を表す条件付き確率を

$$P(y|x) \ (x, y \in \mathbb{F}_2) \tag{13}$$

とする．ここで，x はこの復号法で推定する送信シンボルを表す変数である．確率領域 sum-product 復号法の詳細は次のようになる．

1. **ステップ 1(初期化)**$H_{mn} = 1$ を満たす全ての組 (m, n) に対して $q_{mn}(0) = 1/2$, $q_{mn}(1) = 1/2$ と初期設定する．また，反復回数のカウンタとする変数を $l = 1$ とし，最大反復回数を l_{max} に設定する．
2. **ステップ 2(行処理)**$m = 1, 2, \ldots, M$ の順に $H_{mn} = 1$ を満たす全ての組 (m, n) に対して，次の更新式を利用して $r_{mn}(0)$ と $r_{mn}(1)$ を更新する．

$$r_{mn}(0) = K \sum_{c_i \in F_2, i \in A(m)\backslash n \sum c_i = 0} \left\{ \prod_{n' \in A(m)\backslash n} q_{mn'}(c'_n) P(y'_n | c'_n) \right\} \tag{14}$$

$$r_{mn}(1) = K \sum_{c_i \in F_2, i \in A(m)\backslash n \sum c_i = 1} \left\{ \prod_{n' \in A(m)\backslash n} q_{mn'}(c'_n) P(y'_n | c'_n) \right\} \tag{15}$$

ここで定数 K は，$r_{mn}(0) + r_{mn}(1) = 1$ が成り立つように定められる正規化定数とする．また，表記 $A(m)\backslash n$ は n を除く $A(m)$ に含まれる全ての変数が全ての可能な値 (定義域内のすべての値) をとることを意味する．本来，差集合は $A(m)\backslash\{n\}$ と表記すべきだが，表記を簡単にするためにこの記法を用いるものとする．

3. **ステップ 3(列処理)**$n = 1, 2, \ldots, N$ の順に $H_{mn} = 1$ を満たす全ての組 (m, n) に対して，次の更新式を利用して $q_{mn}(0)$ と $q_{mn}(1)$ を更新する．

$$q_{mn}(0) = K' \prod_{m' \in B(n) \backslash m} r_{m'n}(0) \tag{16}$$

$$q_{mn}(1) = K' \prod_{m' \in B(n) \backslash m} r_{m'n}(1) \tag{17}$$

ここで K' は $q_{mn}(0) + q_{mn}(1) = 1$ となるように定められる正規化定数とする．

4. **ステップ 4(一時推定語の計算)** $n = 1, 2, \ldots, N$ について

$$Q_n(0) = K'' P(y_n | x_n = 0) \prod_{m' \in B(n)} r_{m'n(0)} \tag{18}$$

$$Q_n(1) = K'' P(y_n | x_n = 1) \prod_{m' \in B(n)} r_{m'n(1)} \tag{19}$$

$$\hat{x} = \begin{cases} 0 & \text{if} Q_n(0) \geq Q_n(1) \\ 1 & \text{if} Q_n(0) < Q_n(1) \end{cases}$$

を計算する．ここで得られる $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$ を一時推定語と呼ぶ．定数 K'' は $Q_n(0) + Q_n(1) = 1$ となるように定められる規格化定数である．

5. **ステップ 5(パリティ検査)** 一時推定語が符号語になっているかどうかを検査する．もし $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$ が

$$(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N) H^T = 0 \tag{20}$$

を満たせば，$(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$ を推定語として出力し，アルゴリズムを終了する．

6. **ステップ 6(反復回数のカウント)** もし $l = l_{max}$ ならば，l をインクリメントしてステップ 2 に戻る．$l = l_{max}$ ならば，$(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_N)$ を推定語として出力し，アルゴリズムを終了する．

以上のアルゴリズムを用いた復号シミュレーションを行い，復号誤り確率を検証していく．

3 クラスカル法

クラスカル法は，グラフ理論において重みつき連結グラフの最小全域木を求めるアルゴリズムの 1 つである．本研究では，このアルゴリズムを検査行列の生成に応用した．文献 [5] に基づいて一般的なクラスカル法を説明する．

重みつき連結グラフとは，頂点と辺から構成され，各辺に重みと呼ばれる数値が割り当てられている．最小全域木とは，基のグラフの全ての頂点を含み，選んだ辺にループがなく連結で，選んだ辺の重みの合計が最小である木を指す．クラスカル法の仕組み，手順を説明する．

1. 辺の重みを昇順に 1 つ選ぶ. このとき同じ重みの辺が複数ある場合，どの辺から選んでも構わない.
2. 選ばれた辺を追加したとき，ループが形成されないか確認する.
3. ループが形成されないとき，選んだ辺で頂点を連結する.
4. 全ての辺を選び，確認するまで 2 と 3 を繰り返す.
5. 最終的に全ての頂点が同じ木に属し，選ばれた辺の重みの合計が最小となる最小全域木になる.

実際に具体例を用いて，クラスカル法の手順を説明する．図 2 が初めのグラフとして与えられたとする．図 2 において，分かりやすい説明のために各辺にアルファベットのラベルを付与した.

クラスカル法の手順に従って，図 2 を最小全域木に変換する．まず，各辺を昇順にしたとき最小の重み 1 の辺が a, d, e の 3 つ存在する．同じ重みの辺が複数ある場合，どの辺から選んでも構わないため，辺 a を選び，図 3 になる.

次に，小さい重みの辺も d, e であり，辺 d を選ぶ．辺 d を追加した後，グラフの中でループが形成されないため，追加できる．その結果が図 4 になる.

次に小さい重みの辺は e である．辺 e を追加した後，グラフの中でループが形成されないため，追加できる．その結果が図 5 になる.

次に小さい重みの辺は，重みが 2 の辺 b である．しかし，辺 b を追加した後，グラフの中でループが形成される．したがって，辺 b は最小全域木を構成する候補の辺から除外される．次

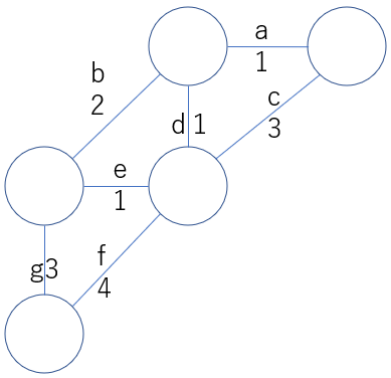

図 2　例としてのグラフ

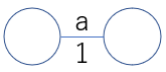

図 3　クラスカル法の過程 1

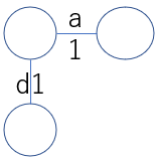

図 4　クラスカル法の過程 2

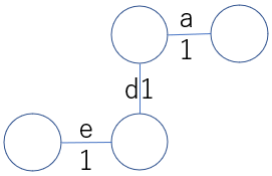

図 5　クラスカル法の過程 3

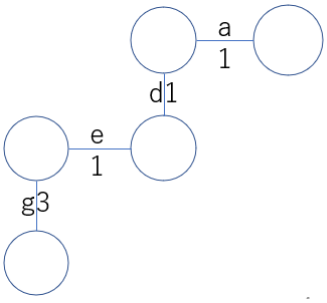

図 6　クラスカル法の過程 4

に小さい重みの辺は，重みが 3 の c, g であり，辺 c を選ぶ．しかし，辺 c を追加した後，グラフの中でループが形成される．したがって，辺 c は最小全域木を構成する候補の辺から除外される．次に，辺 g を選ぶ．辺 g を追加した後，グラフの中でループが形成されないため，追加できる．その結果が図 6 になる．

次に小さい重みの辺は，重みが 4 の辺 f である．しかし，辺 f を追加した後，グラフの中でループが形成される．したがって，辺 f は最小全域木を構成する候補の辺から除外される．これで全ての辺を選び，確認した．よって，図 6 がクラスカル法を用いて作成された図 2 の最小全域木であり，合計の重みは 6 である．

4　クラスカル法による検査行列の生成

この章では，クラスカル法によって検査行列を生成する方法について説明する．

まず，検査行列を生成するために 2 部グラフを導入した．2 部グラフとは，頂点が 2 つの独立した集合に分かれ，異なる集合の頂点同士の間に限って辺が存在するグラフである．2 部グラフを構成するノード（頂点）は 2 つのグループに分かれ，それぞれ M 個と N 個のノードからなる．各グループに属するノードをそれぞれメッセージノードとチェックノードと呼ぶ．この 2 部グラフでは，$M \times N$ 行列の検査行列が与えられる．クラスカル法を適用する前の基となるグラフとして，メッセージノードとチェックノードがそれぞれ全て結ばれた 2 部グラフを考える．そのような 2 部グラフを完全 2 部グラフと呼び，例を図 7 に示す．完全 2 部グラフは，2 部グラフの 1 種である．図 7 のような完全 2 部グラフにクラスカル法を適用し，ループのない 2 部グラフに変換する．このとき，図 7 のグラフには辺に重みがついていない．クラスカル法において辺の重みは，辺を剪定する順番を決めるために用いられている．そのため，辺に重みがついていないグラフに対しても事前に辺の剪定する順番を決めておくことで，クラスカル法を適用することができる．図 7 にクラスカル法を適用したとき，作成できるループのない 2 部グラフの例を図 8 に示す．図 8 のような 2 部グラフを基にして検査行列に変換する．

ループのない 2 部グラフから検査行列への変換方法を説明する．2 部グラフで，j 番目のメッセージノードと i 番目のチェックノードが辺で結ばれていたとき，検査行列の i 行 j 列目要素を 1 とする．これを全ての辺で適応させて，ループのない 2 部グラフに対する検査行列を作成する．よって図 8 のような 2 部グラフでは検査行列は

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{21}$$

となる．このように定義された 2 部グラフでは，j 番目のメッセージノードの次数は検査行列の j 列目の重みに等しく，i 番目のチェックノードの次数は検査行列の i 行目の重みに等しくなる．

検査行列を作成する上で, はじめに検査行列のサイズを指定する．つまり，基となる 2 部グラフのメッセージノードとチェックノードの数を初めに指定する．次にその 2 部グラフのメッセージノードとチェックノードでそれぞれ全て結び，完全 2 部グラフとする．これに対してクラスカル法を適用すれば，ループのない検査行列を作ることができる．しかし，図 7 や図 8 に挙げた例のグラフには辺に重みがついていない．そのようなグラフに対して工夫してクラスカル法を適用する方法を 2 つ考えた．次にそれらを説明する．

4.1 辺に重みをランダムに付与する

この手法では，最初に 2 部グラフの辺の総数に対応する数字を 1 から順に用意する．図 7 の完全 2 部グラフでは，辺の総本数は 15 本であるため，1 から 15 までの数字を用意する．これらの数字をクラスカル法を適用する前に，完全 2 部グラフの各辺にランダムに重みとして付与

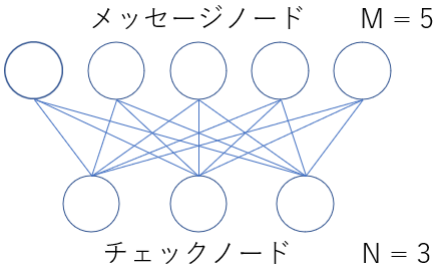

図 7　完全 2 部グラフの例

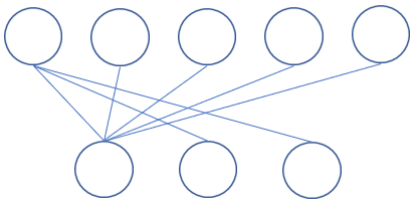

図 8　図 7 をループのない 2 部グラフに変更したときの例

する．同じ重みは使用せず，各辺に異なる重みを付与する．その重みに従って，クラスカル法で完全 2 部グラフの辺が剪定されていく．これによって，各辺に異なるランダムな重みが付与された 2 部グラフとそれに対応した検査行列が作成できる．しかし，この方法ではランダムに重みを付与しているため，辺を剪定する順番もランダムになる．そのため，辺を剪定する順番によって作成できる検査行列に偏りが生じる可能性がある．また, 辺の剪定の順番が作成できる検査行列の種類にどのような影響を及ぼすかが不透明である．この問題に対処するために，初めの完全 2 部グラフから作成できる検査行列を全て求めることを目指した．次にその手法を説明する．

4.2　重みを付与せずに検査行列を網羅的に作成する

完全 2 部グラフの辺にランダムな重みを付けることをやめ，基の完全 2 部グラフから作成できるループのない 2 部グラフを全て求めた．最初に図 7 のような基となる完全 2 部グラフを用意する．次に，辺の総本数の階乗を求める．図 7 の場合，辺の総本数は 15 本であるため，15！を計算する．この計算結果が，完全 2 部グラフの辺を順番に全て選ぶときの場合の数である．つまり，図 7 の完全 2 部グラフでは，辺に重みを付与せずに異なる順序で辺を選ぶ方法が 1307674368000 回（15！回）存在する．毎回選ぶ辺の順序を変更してクラスカル法を適用することで，15！回 2 部グラフを作成する動作ができる．しかし，この過程では全く同じ構造の

2 部グラフが存在する．全く同じ構造であるため，作成できる検査行列も等しくなる．全く同じ構造の 2 部グラフを排除することで，図 7 にクラスカル法を適用したとき作成できる 2 部グラフを全て求めることができる．したがって，図 7 で作成できるループのない 2 部グラフ及びそれに対応する検査行列を全て列挙できた．

5　復号性能の検証

本研究では，ループのないすべての 2×3 検査行列の復号性能を調べた．実際に行った検証方法を説明する．

5.1　検証方法

本研究では，2×3 の完全 2 部グラフからクラスカル法を用いて検査行列を作成した．結果として 12 個の検査行列が得られた．図 9 が得られた 12 個の検査行列である．それぞれの検査行列を基にデコーダと生成行列を作成し，生成行列を基にエンコーダを作成した．2 元対称通信路での符号語の反転確率は，0〜0.5 とした．そのエンコーダ，2 元対称通信路，デコーダに符号語長 3 のメッセージを 10 万回送信した．符号語の復号方法として，sum-product 復号法を用いた．そして，それぞれの検査行列の復号誤り確率を求めることで復号性能を調べた．

$$
\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}
\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}
\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}
\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}
\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}
\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}
\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}
\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}
$$

図 9　ループのない 12 個の検査行列

5.2　検証結果と考察

12 個の検査行列の復号誤り確率を 1 つのグラフにまとめた．図 10 が検証の結果を表すグラフである．図 10 の横軸は通信路での誤り確率，縦軸は検査行列の復号誤り確率を指す．復号誤り確率に大きな差がなかったため，グラフ上では 1 つの線として重なって見えている．12 個の異なる検査行列の中でも復号誤り確率が全く同じ結果となった検査行列がいくつか存在し，5 つのグループに分けることができた．図 11 に示した行列が作成された 12 個の検査行

列である．図 11 において番号でラベル付けされた横に並ぶ各検査行列は全く同じ誤り確率になった．番号で検査行列をグループ分けしている．①〜④のグループの 6 つの検査行列は, 復号誤り確率の悪い検査行列で，差がほとんどなかった．図 10 では，オレンジ線のような推移をとり，性能の悪い検査行列は 1 つに重なって見えている．⑤の 6 つの検査行列は，復号誤り確率が突出して良かった検査行列だった．図 10 では青線で示されている．通信路の確率が 0.4 のとき，性能の悪い検査行列と性能の良い検査行列には，復号誤り確率に約 1.8 倍の差があった．したがって，クラスカル法によって作成した 12 個の検査行列には，復号誤り確率に優劣があることが分かった．本研究では用いた検査行列のサイズが小さいため，他の検査行列と比べて復号誤り確率が突出して良かった原因はまだ分からなかった．

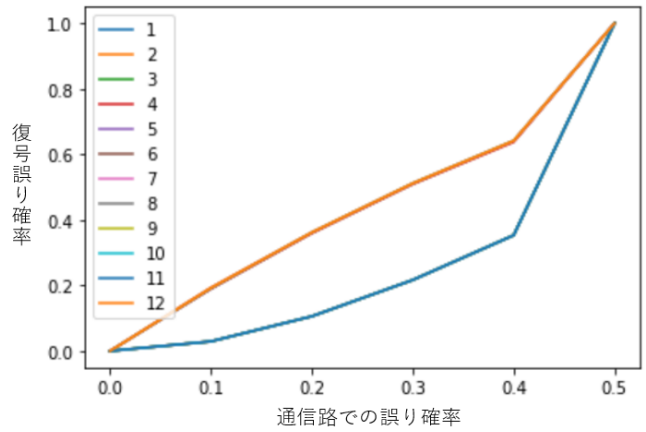

図 10　検査行列の復号誤り確率

①	$\begin{pmatrix}1&1&1\\1&0&0\end{pmatrix}$
②	$\begin{pmatrix}1&0&0\\1&1&1\end{pmatrix}$
③	$\begin{pmatrix}1&1&1\\0&1&0\end{pmatrix}\begin{pmatrix}0&1&0\\1&1&1\end{pmatrix}$
④	$\begin{pmatrix}1&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}0&0&1\\1&1&1\end{pmatrix}$
⑤	$\begin{pmatrix}1&1&0\\1&0&1\end{pmatrix}\begin{pmatrix}1&0&1\\1&1&0\end{pmatrix}\begin{pmatrix}1&1&0\\0&1&1\end{pmatrix}\begin{pmatrix}0&1&1\\1&1&0\end{pmatrix}\begin{pmatrix}0&1&1\\1&0&1\end{pmatrix}\begin{pmatrix}1&0&1\\0&1&1\end{pmatrix}$

図 11　作成された 12 個の検査行列

6　まとめ

本研究では，クラスカル法を用いて完全 2 部グラフから作成できる検査行列を全て作成した．その検査行列を用いてエンコーダ，デコーダを作成し，2 元対称通信路で検査行列の復号誤り確率を調べた．作成した検査行列同士には復号誤り確率に優劣があることが分かった．今後の展望として，検査行列のサイズを大きくして，検査行列の復号誤り確率に優劣が生じた原因を突き止めたい．また，原因を特定することができた後に，復号誤り確率の小さい検査行列を生成する方法を考えたい．2×3 の完全 2 部グラフでは，12 個の検査行列のうち，6 個の検査行列の復号誤り確率が小さく，性能が良かった．検査行列のサイズを大きくしたとき，性能の良い検査行列がクラスカル法によって出力される確率を検証することも検討している．

謝辞

本研究を行うにあたって，丁寧なご指導を賜りました指導教員の西新幹彦准教授に深く感謝申し上げます．また，数多くのご意見をしてくださった西新研究室の方々に深く感謝申し上げます．

参考文献

[1] 和田山正，低密度パリティ検査符号とその復号法，トリケップス， 2002.

[2] 和田山正，誤り訂正技術の基礎，森北出版，2010 年 7 月．

[3] 飯島一貴,「パケット間隔で情報を送る通信路に対する誤り訂正符号の構成に関する考察」，信州大学大学院理工学系研究科修士論文（指導教員：西新幹彦），2017 年 3 月．

[4] 塚田芳寿,「誤り訂正符号としてのラテン方陣の復号方法について」，信州大学大学院総合理工学研究科修士論文（指導教員：西新幹彦），2021 年 3 月．

[5] クラスカル法を python で実装し，アルゴリズムの挙動を見る｜アウトプット雑記，https://output-zakki.com/kruskal_method/, 2023 年 1 月閲覧．

[6] C 言語による乱数生成，https://omitakahiro.github.io/random/random_variables_generation.html, 2023 年 10 月閲覧．

付録 A　完全 2 部グラフに対してクラスカル法を適用するプログラム

文献 [5] を参考にして，プログラムを作成した．

```
def kruskal(Edges, V):
    """
    入力
    Edges: (weight, (From_node, To_node)) を管理するリスト
    V: 頂点数

    出力
    ans: 最小全域木の重み
    """

    test = []
    uf = UnionFind(V) #UnionFind を初期化．初期は全頂点が異なるグループ
    for (From, To) in Edges:

        #同じグループ（木）に属する場合，その辺は無視する
        if uf.isSame(From, To):
            continue
        #異なるグループの場合，その辺を採用し，頂点 From，To を同じグループにする
        else:
            uf.unite(From, To)
            #print((From, To)) #採用されたエッジを出力
            test.insert(len(test), (From, To)) #採用されたエッジをリスト　test に格納

    return test

#=======以下,Union-Find 木 各頂点の管理に利用===============
class UnionFind():
    def __init__(self, n):
        self.n = n
        self.parents = [-1]*n
        self.size = [1]*n
    def root(self, x):
        if self.parents[x] == -1:
            return x
        else:
            self.parents[x] = self.root(self.parents[x])
            return self.parents[x]
    def isSame(self, x, y):
        if self.root(x)==self.root(y):
            return True
        else:
            return False
    def unite(self, x, y):
        x = self.root(x)
        y = self.root(y)
        if x==y:
            return
        self.parents[y] = x #y の根を x に変更


#（出発ノード，行き先ノード）として管理
Edges = [(0, 3), (0, 4),
         (1, 3), (1, 4),
         (2, 3), (2, 4)]


Edges2 = []

#以下から Edges を全通りで並び替え（6!）
for i in range(720):
```

```
    a = divmod(i,6)
    b = divmod(a[0],5)
    c = divmod(b[0],4)
    d = divmod(c[0],3)
    e = divmod(d[0],2)

    sample = [(0, 3), (0, 4),
              (1, 3), (1, 4),
              (2, 3), (2, 4)]

    sample.insert(len(sample), sample[a[1]])
    sample.remove(sample[a[1]])

    sample.insert(len(sample)-1, sample[b[1]])
    sample.remove(sample[b[1]])

    sample.insert(len(sample)-2, sample[c[1]])
    sample.remove(sample[c[1]])

    sample.insert(len(sample)-3, sample[d[1]])
    sample.remove(sample[d[1]])

    sample.insert(len(sample)-4, sample[e[1]])
    sample.remove(sample[e[1]])

    Edges2.insert(0, kruskal(sample, len(sample)))

    #print(Edges2)

    for n in range(1, len(Edges2)):
        if set(Edges2[0]) == set(Edges2[n]):
            Edges2.remove(Edges2[0])#1 つ削除
            break

print(Edges2)
print(len(Edges2))
```

付録 B　sum-product 復号法を用いたデコーダで復号誤り確率を求めるプログラム

文献 [3]，文献 [6] を参考にして，プログラムを作成した．

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include "MT.h"
#define ROW 2    //行
#define COLUMN 3 //列
#define LOOP_SUM_PRODUCT 3 /*sum-product 復号の反復回数設定*/
#define FLIP 0.3 //1 と 0 の反転確率
#define KAISU 100000

double rfx[2][ROW][COLUMN];
double qxf[2][ROW][COLUMN];
double g[2];
double cc[2];
double ss[2];
double gg[2];



double w[2][2] = { 0.7, 0.3, 0.3, 0.7 };
```

```
int parity[ROW][COLUMN];


void decoder(int y[], int x_hat[])
{
    int i, j, n;
    int k, m, l, q;
    int length;
    int sum;
    int max1;
    //double max2;
    //int likely_word;
    double check_sum[2];
    int index[COLUMN];
    double normal_const;
    //int x_hat[COLUMN];



    //検査行列の 1 がある部分に初期値を入れる
    for (i = 0; i < ROW; i++) {
        for (j = 0; j < COLUMN; j++) {
            for (l = 0; l < 2; l++) {
                rfx[l][i][j] = 0.0;
                if (parity[i][j] == 0) {
                    qxf[l][i][j] = 0;
                    //printf("%f",qxf[l][i][j]);
                }
                else {
                    qxf[l][i][j] = 1.0 / 2.0;
                    //printf("%f",qxf[l][i][j]);
                    //printf("%d,   %d\n", i ,j);
                }
                //printf("%f", qxf[l][i][j]);
            }
        }

    }

    for (int loop = 0; loop < LOOP_SUM_PRODUCT; loop++) {
        //r の更新 (M f → x )
        for (i = 0; i < ROW; i++) {
            for (j = 0; j < COLUMN; j++) {
                //検査行列が 1 であるところを探す (列ごと)
                if (parity[i][j] == 0) continue;

                for (l = 0; l < 2; l++) {
                    check_sum[l] = 0.0;
                }


                length = 0;

                //index には列中 1 の位置を格納 (今見ている 1 以外)
                //length は列中の 1 の個数
                for (q = 0; q < COLUMN; q++) {
                    if ((q == j) || (parity[i][q] == 0)) continue;
                    index[length] = q;
                    length++;
                }

                //繰返し回数を決める
                max1 = 1 << length;  //全通りの回数の設定  ビットシフト
                //printf("%d\n", max1);

                for (m = 0; m < max1; m++) {
                    for (l = 0; l < 2; l++) {
```

```
                ss[l] = 1.0;
            }
            for (l = 0; l < 2; l++) {
                sum = 0;
                for (n = 0; n < length; n++) {
                    sum ^= (m >> n) & 1;
                }
                if ((sum ^ l) != 0) continue;
                for (n = 0; n < length; n++) {
                    ss[l]
                        *=
                        qxf[(m >> n) & 1][i][index[n]] *
                        w[y[index[n]]][(m >> n) & 1];

                    //printf("%f\n", ss[l]);
                }
                check_sum[l] += ss[l];
                //printf("%f", check_sum[l]);

            }

        }

        normal_const = 0.0;
        for (l = 0; l < 2; l++) {
            normal_const += check_sum[l];
        }
        //printf("%f", check_sum[l]);
        for (l = 0; l < 2; l++) {
            rfx[l][i][j] = check_sum[l] / normal_const;
            //printf("%f", rfx[l][i][j]);
        }

    }
}



for (j = 0; j < COLUMN; j++) {
    for (i = 0; i < ROW; i++) {
        if (parity[i][j] == 0) continue;
        for (l = 0; l < 2; l++) {
            cc[l] = 1.0;
        }

        for (k = 0; k < ROW; k++) {
            if ((k == i) || (parity[k][j] == 0)) continue;
            for (l = 0; l < 2; l++) {
                cc[l] *= rfx[l][k][j];
            }
        }

        normal_const = 0.0;

        for (l = 0; l < 2; l++) {
            normal_const += cc[l];
        }

        for (l = 0; l < 2; l++) {
            qxf[l][i][j] = cc[l] / normal_const;
        }
    }
}

for (j = 0; j < COLUMN; j++) {
    for (l = 0; l < 2; l++) {
        gg[l] = 1.0;
        for (i = 0; i < ROW; i++) {
            if (parity[i][j] == 0) continue;
```

```
                    gg[l] *= rfx[l][i][j];
                }
                gg[l] *= w[y[j]][l];
            }

            normal_const = 0.0;

            for (l = 0; l < 2; l++) {
                normal_const += gg[l];
            }


            for (l = 0; l < 2; l++) {
                g[l] = gg[l] / normal_const;
                //printf("%f\n", g[l]);
            }

            x_hat[j] = (g[0] > g[1]) ? 0 : 1; //尤度の比較
            //printf("%d\n", x_hat[j]);
            //putchar('\n');

        }


    }
    return;
}

int transmit(int x[], int y[])
{
    for (int i = 0; i < COLUMN; i++) {
        if (genrand_real2() < FLIP) {
            y[i] = 1 - x[i]; //反転
        }
        else {
            y[i] = x[i];
        }
    }
    return(0);
}

double decoding_error()
{
    int word[COLUMN] = { 0,0,0 };
    int x[COLUMN]; //送信される符号語
    int y[COLUMN]; //受信語
    int x_hat[COLUMN]; //復号語
    int count = 0;

    for (int i = 0; i < KAISU; i++) {
        int j;
        for (j = 0; j < COLUMN; j++) {
            x[j] = word[j];
        }
        transmit(x, y);
        decoder(y, x_hat);

       //printf("%d%d%d -> %d%d%d -> %d%d%d\n", x[0], x[1], x[2],y[0], y[1], y[2], x_hat[0], x_hat[1], x_hat[2]);

        for (j = 0; j < COLUMN; j++) {
            if (x_hat[j] != x[j]) {
                break;
            }
        }
        if (j < COLUMN) {
            count++;
            //printf("error\n");
        } /*else {
            printf("ok\n");
```

```
        }*/
    }
    printf("count: %d\n", count);

    return((double)count / KAISU);
}


int main(void) {
    int i, j;
    int row, column;
    double error;

    init_genrand(10);

    for (i = 0; i < ROW; i++) {
        for (j = 0; j < COLUMN; j++) {
            parity[i][j] = 0;
        }
    }

    FILE* fp;

    if ((fp = fopen("limit_test.txt", "r")) == NULL) {
        printf("\a no open\n");
        exit(1);
    }
    else {
        while (fscanf(fp, "%d %d", &column, &row) != EOF) {
            parity[row - COLUMN][column] = 1;
        }
    }
    fclose(fp);

    error = decoding_error();

    printf("error: %g\n", error);

    return 0;

}
```