信州大学
大学院総合理工学研究科

修士論文

制約付き乱数に基づく情報源符号化の性能検証に向けて

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　23W2092A
氏名　南澤 航

2025 年 2 月 7 日

目次

1　はじめに

通信を行う際，送信元から受信先に正しくメッセージ（情報）を伝えられない場合がある．これは，途中経路である通信路において雑音が発生し，本来伝えようとしていた情報に誤りが生じてしまうためである．このため，発生した誤りを検出し正しく復号する，誤り訂正技術は雑音のある環境下において必要不可欠である．

Shannon は文献 [1] において，雑音のある通信路が与えられた場合，その通信路において，メッセージを正しく送ることができる効率の上限を示した．この上限は通信路容量と呼ばれ，通信路容量を達成するような符号は文献 [1] ですでに提案されている．しかし，この符号では計算時間，メモリ量が指数的に増大するため，現実的な実行は困難である．このため，通信路容量を達成するような実行可能な符号を考えることは，情報理論における課題とされてきた．

従来の符号技術では，実装の容易さから線形符号と呼ばれるものが用いられ，これらの符号の一部は通信路容量に近い性能を示すことが確認されている．しかし，線形符号が通信路容量を達成するのは，線形符号の性質上，特定の通信路に限られており一般には通信路容量を達成するとは限らない．そこで，一般の通信路に対して通信路容量を達成する符号技術として文献 [2] において CoCoNuTS(Code based on Constrained Numbers Theoretically-achieving the Shannon limit，拘束条件を満たす系列に基づくシャノン限界を達成する符号）が提案された．

CoCoNuTS は，特殊な乱数生成器を使用することによって，あらゆる通信路で通信路容量を達成することができる通信路符号化の技術である．そして，その符号器，復号器では乱数を用いた情報源符号化の考えが用いられている．筆者は文献 [3] で CoCoNuTS による通信路符号化を行い，送受信シミュレーションを通してその性能を検証した．しかし、符号のサイズか小さく，さらに特定の条件でのみ検証を行ったため，CoCoNuTS が持つ性能を十分に調べることができなかった．

そこで，本研究では CoCoNuTS による通信路符号化ではなく，その要素技術である乱数生成器を用いた情報源符号化に着目することで，CoCoNuTS の持つ性能を詳細に調べた．さらに，復号方法を文献 [3] で使用したブロック MAP 復号法から，Sum-Product アルゴリズムを利用した復号方法に変えることによって，大小様々なサイズの符号で検証を行った．

2　制約付き乱数生成器による情報源符号化

この章ではまず，通信路符号化問題についてその概要を述べる．そして，CoCoNuTS による通信路符号化について簡単に説明する．その後，CoCoNuTS の要素技術であり本研究のテーマである，制約付き乱数生成器による情報源符号化について文献 [2] を元に説明する．

図 1　通信路モデル

2.1　通信路符号化問題

図 1 は一般的な通信路符号化のモデルである．ここで，通信路とは現実の通信システムを数学的にモデル化したものであり，通信路入出力 X と Y との間の条件付き確率分布 $P_{Y|X}$ で表される．また，通信路入力 X と通信路出力 Y の分布はそれぞれ P_X，P_Y と表される．これらの確率は，通信システムを考える上で重要であり，符号器や復号器の設計と深く関係する．以下では，このような確率モデルとして与えられた通信路に対して，符号の性能がどのように表されるか述べていく．

図 1 において，符号器では送信者から受け取ったメッセージ M を符号化して通信路入力 X へ変換する．変換された X は通信路へと入力され，復号器は通信路出力 Y を得る．ここで，通信路において発生した雑音の影響により，通信路入出力 X と Y において $X \neq Y$ となるような場合が考えられる．このため，復号器では観測された Y をもとに送信メッセージ M の推定を行い，推定値を M' として出力する．ここで，このようにして出力された M' は，送信者が本来送りたかったメッセージ M とは異なる場合が考えられる．このように $M \neq M'$ となるような確率は復号誤り確率と呼ばれ [4]，符号の性能を測る尺度の一つである．

次に通信を行う上での効率を測る尺度である，符号化レートについて説明する．符号語長を n，メッセージ数を M_n とした場合，符号化レート R は

$$R = \frac{\log_2 M_n}{n} \tag{1}$$

と表される [4]．符号化レートが大きいほど通信が高速であることを意味する．

通信を行う上で性能の良い符号とは，符号化レートが可能な限り大きく，同時に復号誤り確率が限りなく 0 に近い符号である．ここで，復号誤り確率を 0 に近づけながら，符号化レートを大きくするとき，大きく出来る符号化レートの上限は，通信路ごとに決まる．このような符号化レートの限界を通信路容量とよび，文献 [5] で次のように定義されている．

定義 1　与えられた通信路に対して定まる値 C が以下の二つの条件を満たすとき，C をこの

通信路の通信路容量と呼ぶ.

- 復号誤り確率が 0 に限りなく近く，かつ符号化レート R が C に限りなく近い符号が存在する.
- 復号誤り確率が 0 に限りなく近く，かつ符号化レート R が $R > C$ を満たす符号は存在しない.

定義 1 のもとで次の定理が成り立つ.

定理 1　通信路 $P_{Y|X}$ の通信路容量 $C(P_{Y|X})$ は

$$C(P_{Y|X}) = \max_{P_X}(H(X) - H(X|Y)) \tag{2}$$

で表される．ここで，$H(X)$ は X のエントロピー，$H(X|Y)$ は Y が与えられた元での X の条件付きエントロピーと呼ばれ，それぞれ

$$H(X) \triangleq -\sum_x P_X(x) \log P_X(x) \tag{3}$$

$$H(X|Y) \triangleq -\sum_x \sum_y P_{XY}(x,y) \log P_{X|Y}(x|y) \tag{4}$$

と定義される．また，式 (2) の右辺に現われる，$H(X) - H(X|Y)$ は相互情報量 $I(X;Y)$ を用いて

$$C(P_{Y|X}) = \max_{P_X}(I(X;Y)) \tag{5}$$

と置き換えることが出来る.

式 (5) から通信路容量とは相互情報量 $I(X;Y)$ の最大値であるということがわかる．このとき，式 (5) の最大値を与えるような通信路の入力分布 P_X は，最適な入力分布と呼ばれる．またこのとき，入力分布 P_X は通信路容量を達成するという．最適な入力分布は通信路ごとに異なり，性能の良い符号を設計する上で重要となる.

通信路符号化の問題では通常，送信されるメッセージは一様分布に従うことが仮定される．このとき，従来法として知られている線形符号では，メッセージと通信路入力が一対一に対応する特性があり，通信路入力である X の各シンボルも一様分布に従う．よって，線形符号により通信路容量が達成されるのは，最適な入力分布が一様分布となるような通信路だけであり [6][7]，通信路入力が一様分布のときに通信路容量を達成できない通信路では，原理上，通信路容量を達成できない [8].

線形符号のこのような問題を解決するため，CoCoNuTS では符号器，復号器に特殊な乱数生成器を使用している．次節では CoCoNuTS の通信モデルを用いて，CoCoNuTS が通信路容量を達成する原理を説明する.

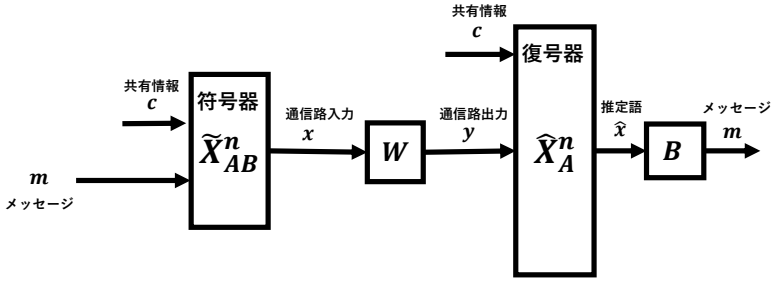

図 2　CoCoNuTS の通信路モデル

2.2　CoCoNuTS による通信路符号化

図 2 は CoCoNuTS による通信路符号化のモデルである．以下では $\mathcal{X}$ を有限体とし，有限体上での計算を進める．このとき，符号器では，送信者から受け取ったメッセージ $\boldsymbol{m} = (m_1, \cdots, m_k)$ と，符号器，復号器に与えられる共有情報 $\boldsymbol{c} = (c_1, \cdots, c_l)$ から分布

$$\widetilde{X}_{AB}^{n}(\boldsymbol{x}|\boldsymbol{c}, \boldsymbol{m}) = \frac{P(\boldsymbol{x})\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}, B\boldsymbol{x} = \boldsymbol{m}\}}{\sum_{\boldsymbol{x}} P(\boldsymbol{x})\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}, B\boldsymbol{x} = \boldsymbol{m}\}} \tag{6}$$

に従って通信路入力 $\boldsymbol{x} = (x_1, \cdots, x_n)$ を生成する．ここで，A は $l \times n$ 行列，B は $k \times n$ 行列であり，$\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}, B\boldsymbol{x} = \boldsymbol{m}\}$ は定義関数

$$\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}, B\boldsymbol{x} = \boldsymbol{m}\} \triangleq \begin{cases} 1 & A\boldsymbol{x} = \boldsymbol{c}, B\boldsymbol{x} = \boldsymbol{m} \text{ を満たしているとき} \\ 0 & A\boldsymbol{x} = \boldsymbol{c}, B\boldsymbol{x} = \boldsymbol{m} \text{ を満たしていないとき} \end{cases} \tag{7}$$

を表している．また，$\sum_{\boldsymbol{x}}$ はすべての n 次元列ベクトル $\boldsymbol{x}$ にわたる和である．さらに，拘束条件 $A\boldsymbol{x} = \boldsymbol{c}, B\boldsymbol{x} = \boldsymbol{m}$ を満たすような解 $\boldsymbol{x}$ が複数あることを保証するため $l + k \leq n$ を仮定する．

式 (6) は拘束条件を満たす系列 $\boldsymbol{x}$ を与えられた確率 $P(\boldsymbol{x})$ に比例してランダムに生成するものであり，拘束条件を満たす乱数生成器と呼ばれている．

復号器でも符号器と同様に，拘束条件を満たす乱数生成器が用いられている．復号器では通信路出力 $\boldsymbol{y} = (y_1, \cdots, y_n)$ と共有情報 $\boldsymbol{c}$ を受け取り分布

$$\widehat{X}_{A}^{n}(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{c}) = \frac{P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\}}{\sum_{\boldsymbol{x}} P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\}} \tag{8}$$

に従って通信路入力の推定値 $\hat{\boldsymbol{x}}$ をランダムに生成する．このとき，本来の通信路入力 $\boldsymbol{x}$ が拘束条件 $B\boldsymbol{x} = \boldsymbol{m}$ を満たすことを利用すると，生成した推定符号語 $\hat{\boldsymbol{x}}$ が通信路入力 $\boldsymbol{x}$ と等しい場合，$B\hat{\boldsymbol{x}} = \boldsymbol{m}$ を計算することで，メッセージ $\boldsymbol{m}$ を正しく復元することができる．上記のような乱数生成器を用いて CoCoNuTS の符号器，復号器は構成される．

ここで，CoCoNuTS の符号器では，線形符号と異なり，あるメッセージ $\boldsymbol{m}$ に対して生成される符号語 $\boldsymbol{x}$ の候補が複数存在する．さらに，それらの候補は式 (6) に従ってランダムに選択される．このことにより，通信路入力 $\boldsymbol{x}$ が従う確率分布 P_X の選択が自由になる．従って式 (2) の右辺を最大化するような入力分布を用いて，符号を構成することで，CoCoNuTS は通信路容量を達成することができる．

文献 [2] では，上記のような乱数を用いた符号器，復号器が正しく符号化，復号化を行えることを情報源符号化の考えを元に示している．本研究では，この情報源符号化の考えを元にした符号器，復号器に着目したため，次節ではその点についてより詳細に述べる．

2.3 通信路符号化と情報源符号化

まず，式 (6) で表される符号器 $\widetilde{X}_{AB}^{n}$ が正しく符号化を行えるか調べるために，図 2 の符号器部分について，符号器の動作自体を変えることなく，その解釈を変える．

図 3 は解釈を変えるために，仮の情報源 $\widetilde{X}$ と行列 A, B を追加した，図 2 の符号器部分である．式 (6) で表されるように，符号器では与えられた共有情報 $\boldsymbol{c}$ と，メッセージ $\boldsymbol{m}$ を受け取り，式 (6) の分布に従って，拘束条件 $A\boldsymbol{x} = \boldsymbol{c}$, $B\boldsymbol{x} = \boldsymbol{m}$ を満たすような $\boldsymbol{x}$ をランダムに生成する．

このとき，仮の情報源 $\widetilde{X}$ を追加することによって，共有情報 $\boldsymbol{c}$ と，メッセージ $\boldsymbol{m}$ は，与えられたものではなく，仮の情報源からでた系列 $\tilde{\boldsymbol{x}}$ を行列 A と B で符号化したものだと考えることができる．そして，このように生成された $\boldsymbol{c}$ と $\boldsymbol{m}$ は誤りなく $\widetilde{X}_{AB}^{n}$ に届き $\boldsymbol{x}$ を生成す

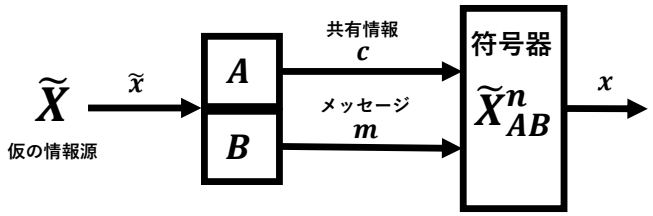

図 3　仮の情報源 $\widetilde{X}$ を追加した CoCoNuTS 符号器のモデル

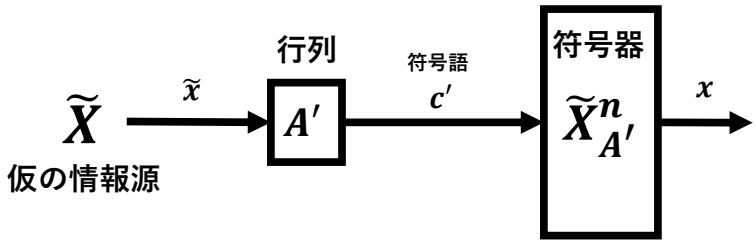

図 4　行列 A と B を結合させたモデル

る．すると，図 3 で表されるモデルは，行列 A, B のペアを符号器，$\widetilde{X}^n_{AB}$ を復号器とした情報源符号化問題であると解釈することができる．すなわち，CoCoNuTS による通信路符号化の符号器 $\widetilde{X}^n_{AB}$ が正しく符号化を行えるということは，このモデルにおいて $\tilde{\boldsymbol{x}}$ と $\boldsymbol{x}$ が等しくなることに相当する．

上記のような解釈を行うことによって，通信路符号化の符号器 $\widetilde{X}^n_{AB}$ の動作を情報源符号化問題として，考えることができる．なお，前節で述べたように行列 A と B は列の長さが等しい行列である．そのため，$l \times n$ 行列 A と $k \times n$ 行列 B で，共有情報 $\boldsymbol{c}$ と，メッセージ $\boldsymbol{m}$ を符号化する図 3 のモデルは，行列 A と B を縦に結合させた $(l+k) \times n$ 行列 A' で符号語 $\boldsymbol{c'}$ を生成する，図 4 のようなモデルと考えることもできる．

このように考えることによって，図 3 の情報源符号化問題をよりシンプルに捉えることができる．

次に式 (8) で表される復号器 $\widehat{X}^n_A$ についても，正しく復号が行われるか調べるために，復号器自体の動作を変えることなく，その解釈を変える．

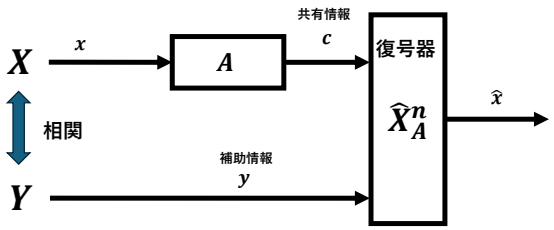

図 5　行列 A を加えた CoCoNuTS 復号器のモデル

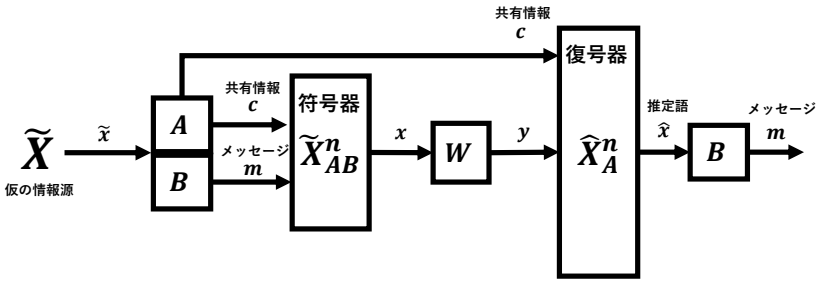

図 6　書き直した CoCoNuTS の通信路モデル

図 5 は，図 2 の復号器部分を切り取り，行列 A を加えたものである．復号器 $\widehat{X}^n_A$ で受け取る共有情報 $\boldsymbol{c}$ は，系列 $\boldsymbol{x}$ と行列 A との積 $A\boldsymbol{x} = \boldsymbol{c}$ を満たすものであるから，行列 A を追加することによって，共有情報 $\boldsymbol{c}$ は与えられるものでなく，行列 A を用いて生成された，と考えることができる．

このとき，図 5 のモデルは，追加した行列 A を符号器，$\widehat{X}^n_A$ を復号器とし，通信路出力 $\boldsymbol{y}$ を復号器 $\widehat{X}^n_A$ の復号を手助けする補助情報とした，補助情報付き情報源符号化問題と解釈することができる．すると，CoCoNuTS による通信路符号化の復号器 $\widehat{X}^n_A$ が正しく復号を行えるということは，このモデルにおいて $\boldsymbol{x}$ と $\hat{\boldsymbol{x}}$ が等しくなることに相当する．このように解釈することで，符号器と同様，通信路符号化の復号器の動作を，補助情報付き情報源符号化問題として考えることができる．

そして，上記のように解釈することによって，CoCoNuTS による通信路符号化のモデルを図 2 から図 6 へと書き直すことができる．

このようにして，CoCoNuTS による通信路符号化の符号器，復号器の動作を，情報源符号化問題として解釈することができた．そして，文献 [2] ではこの情報源符号化問題の定理を根拠として，符号器，復号器の性能を保証している．そのため，次節では文献 [2] をもとに性能保証の原理について説明する．

2.4　制約付き乱数による情報源符号化

前節では，図 4 で表される情報源符号化問題と，図 5 で表される補助情報付き情報源符号化問題について述べた．以下ではまず，補助情報付き情報源符号化問題について文献 [2] をもとに，性能保証がどのようになされているか述べる．その後，補助情報無しの情報源符号化についても，述べる．

まず，符号の構成について説明する．X^n のアルファベット $\mathcal{X}^n$ は有限であり，Y^n のアルファベット $\mathcal{Y}^n$ は任意の集合であると仮定する．さらに，$l \times n$ 行列 A と，その集合 $\mathcal{A}_n$ に対して，$\mathrm{Im}\, A$ と $\mathrm{Im}\, \mathcal{A}_n$ を

$$\mathrm{Im}\, A \equiv \{A\boldsymbol{x} : \boldsymbol{x} \in \mathcal{X}^n\} \tag{9}$$

$$\mathrm{Im}\, \mathcal{A}_n \equiv \bigcup_{A \in \mathcal{A}_n} \mathrm{Im}\, A \tag{10}$$

と定義する．また，ランダムに決定される行列 A に対応する，確率変数をサンセリフ文字 A で表す．そして，$\mathcal{A}_n$ を $\mathcal{X}$ 上の行列の集合，$p_{\mathsf{A},n}$ を $\mathcal{A}_n$ 上の確率分布としたとき，$\mathcal{A}_n$ と $p_{\mathsf{A},n}$ の組 $(\mathcal{A}_n, p_{\mathsf{A},n})$ をアンサンブルと呼ぶことにする．

このとき，集合 $\mathcal{X}^n$ 上の関数のアンサンブル $(\mathcal{A}_n, p_{\mathsf{A},n})$ が，与えられた符号化レート r に対して

$$r = \frac{1}{n} \log |\,\mathrm{Im}\, \mathcal{A}_n| \tag{11}$$

を満たすと仮定する．また，符号化関数 $A : \mathcal{X}^n \to \mathrm{Im}\, \mathcal{A}_n$ を分布 $p_{\mathsf{A},n}$ にしたがってランダムに決定する．このとき，$\boldsymbol{x} \in \mathcal{X}^n$ の符号語 $\boldsymbol{c}$ は $\boldsymbol{c} \equiv A\boldsymbol{x}$ で与えられる．

制約付き乱数生成器を用いて確率的復号器 $\widehat{X}_A^n : \mathrm{Im}\, A \times \mathcal{Y}^n \to \mathcal{X}^n$ を構成する．$C_n \equiv AX^n$ とすると，復号器では，与えられた符号語 $\boldsymbol{c} \in \mathrm{Im}\, A$ と補助情報 $\boldsymbol{y} \in \mathcal{Y}^n$ を用いて，推定語 $\widehat{X}^n \equiv \widehat{X}_A^n(\boldsymbol{c}, \boldsymbol{y}) \in \mathcal{X}^n$ を分布

$$P_{\widehat{X}^n|C_nY^n}(\hat{\boldsymbol{x}}|\boldsymbol{c}, \boldsymbol{y}) = \frac{P_{X^n|Y^n}(\hat{\boldsymbol{x}}|\boldsymbol{y})\mathbb{1}\{A\hat{\boldsymbol{x}} = \boldsymbol{c}\}}{\sum_{\boldsymbol{x}} P_{X^n|Y^n}(\boldsymbol{x}|\boldsymbol{y})\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\}} \tag{12}$$

に従ってランダムに決定する．つまり，この制約付き乱数生成器は $A\hat{\boldsymbol{x}} = \boldsymbol{c}$ を満たす $\hat{\boldsymbol{x}}$ を $P_{\widehat{X}^n|C_nY^n}(\hat{\boldsymbol{x}}|\boldsymbol{c}, \boldsymbol{y})$ の確率で生成する．

そして，このような乱数を用いた復号器の復号誤り確率 Error(A) は

$$\mathrm{Error(A)} = \sum_{\substack{\boldsymbol{x}, \boldsymbol{y}, \hat{\boldsymbol{x}} \\ \boldsymbol{x} \neq \hat{\boldsymbol{x}}}} P_{\widehat{X}^n|C_nY^n}(\hat{\boldsymbol{x}}|A\boldsymbol{x}, \boldsymbol{y}) P_{X^nY^n}(\boldsymbol{x}, \boldsymbol{y}) \tag{13}$$

と表される．

上記のように構成された復号器と，定義された誤り確率を用いて文献 [2] では以下の定理が示されている．

定理 2 X を情報源，Y を補助情報源とし，(X, Y) には相関があると仮定する．アンサンブル $(\mathcal{A}_n, p_{\mathsf{A},n})$ が与えられた符号化レート r に対して，式 (11) と

$$r > \overline{H}(X|Y) \tag{14}$$

を満たすような $(\alpha_{\mathsf{A}_n}, \beta_{\mathsf{A}_n})$−Collision-Resistance 特性を持つことを仮定する．

このとき任意の $\delta > 0$ と十分大きな n に対して，

$$\mathrm{Error(A)} \leq \delta \tag{15}$$

となるような行列 $A \in \mathcal{A}_n$ が存在する．

上記の定理 2 に現われる $(\alpha_{\mathsf{A}_n}, \beta_{\mathsf{A}_n})-$Collision-Resistance 特性とは文献 [2] で定義されるハッシュ特性の変形であり，本研究で扱う符号はこの特性を持つ．

このように，CoCoNuTS による通信路符号化の復号器の動作を，補助情報付き情報源符号化の問題と考えることによって，送信した情報を正しく復元できるような行列 A の存在が，定理 2 から保証される．

また，図 4 で表される補助情報なしの情報源符号化問題についても，上記の手法とほぼ同様の考え方を用いることができる．

前節の図 4 と図 5 を見比べると，この二つのモデルの相違点は，補助情報 $\boldsymbol{y}$ を受け取るかどうかであることが分かる．このため，図 4 の情報源符号化問題は，図 5 の補助情報付き問題における，補助情報 $\boldsymbol{y}$ を受け取らない，特別な場合であると考えることができる．

このことは，数式からも確認できる．前節で述べたとおり，図 4 のモデルは，CoCoNuTS による通信路符号化における符号器の動作を，情報源符号化問題として解釈したものである．そのため，式 (6) より，この情報源符号化問題の復号器 $\widehat{X}^n_{A'}$ は $\boldsymbol{x}$ を，分布

$$P_{X^n|C'_n}(\boldsymbol{x}|\boldsymbol{c'}) = \frac{P_{X^n}(\boldsymbol{x})\mathbb{1}\{A'\boldsymbol{x} = \boldsymbol{c'}\}}{\sum_{\boldsymbol{x}} P_{X^n}(\boldsymbol{x})\mathbb{1}\{A'\boldsymbol{x} = \boldsymbol{c'}\}} \tag{16}$$

に従ってランダムに決定する．ここで，式 (12) と式 (16) を比較すると，式 (16) を用いる図 4 のモデルは，補助情報付き問題における，補助情報 $\boldsymbol{y}$ を受け取らない特別な場合であることが明らかである．

したがって，図 4 で表される補助情報なしの情報源符号化問題に対しても，上記の定理 2 が適用可能である．すなわち，CoCoNuTS の符号器は CoCoNuTS の復号器と同様に，符号化したい情報を正しく符号化できるような行列 A' の存在が保証される．

このように CoCoNuTS では，その符号器，復号器に情報源符号化の考えが用いられている．本研究ではこの情報源符号化に着目しその中でも特に，式 (16) で表される，CoCoNuTS の符号器における情報源符号化について，その性能検証を行った．ここで実際に性能検証を行うにあたり，考慮する点が 2 つある．

1 つ目は，符号化で使用する行列 A' の生成方法である．定理 2 より，性能の良い行列の存在は保証されているが，その候補は膨大である．そのため，効率よく行列生成を行い，性能の良いものの調査を行う必要がある．

2 つ目は復号時の計算量である．復号には式 (16) の計算が必要であるが，これは高次元の確率分布であり直接計算することは困難である．そこで本研究では，このような確率分布の近

似計算を，効率的に行うアルゴリズムとして知られている，Sum-Product アルゴリズムを導入することで計算を行った．次章では，式 (16) をもとに，Sum-Product アルゴリズムについて詳細に述べる．そして，その後行列生成の方法についても述べる．

なお，図 5 で表される補助情報付き情報源符号化問題については，本研究では性能検証を行っていないが，その計算過程について参考として付録 A.1 に記載する．また，図 5 では復号器のみに補助情報が提供されているが，一般的な問題設定では符号器にも補助情報が提供される．そのため，符号器にも補助情報を与える場合の計算過程も付録 A.2 に記載する．

3 Sum-Product アルゴリズム

本研究では，前章で述べた式 (16) に対して Sum-Product アルゴリズムを適用することによって，その近似計算を可能とした．そこで本章では，具体例を用いることで，式 (16) に対する計算がどのようになされているか，詳細に述べる．

3.1 シンボル単位事後確率

式 (16) に対する Sum-Product アルゴリズムの計算手順について，具体的な符号を用いて説明する．

本研究では使用する情報源を，定常無記憶情報源とした．このため，情報源から長さ 6 の系列 $\boldsymbol{x} = (x_1, \cdots, x_6)$ が送信されるとき，その出現確率 $P\{X^6 = \boldsymbol{x}\}$ は

$$P\{X^6 = \boldsymbol{x}\} = \prod_{i=1,2,3,4,5,6} P\{X_i = x_i\} \tag{17}$$

と表せる．

次に，この情報源系列 $\boldsymbol{x}$ を符号化する行列について述べる．本節では具体例として 6×3 行列 A を使用し，この行列 A を

$$A \triangleq \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{18}$$

のように定義する．情報源符号化では，この行列 A と情報源系列 $\boldsymbol{x}$ との積を取ることで符号語 $\boldsymbol{c} = (c_1, \cdots, c_3)$ を生成する．そして，この符号語 $\boldsymbol{c}$ は誤りなく復号器へ届く．このため，$\boldsymbol{c}$ は条件

$$c_1 = x_1 + x_2 + x_3 \tag{19}$$
$$c_2 = x_3 + x_4 \tag{20}$$
$$c_3 = x_4 + x_5 + x_6 \tag{21}$$
$$\tag{22}$$

を満たしていることになる．ここで，今後の計算のために，これら条件を

$$f_1(x_1, x_2, x_3, c_1) \triangleq \mathbb{1}\{x_1 + x_2 + x_3 = c_1\} \tag{23}$$

$$f_2(x_3, x_4, c_2) \triangleq \mathbb{1}\{x_3 + x_4 = c_2\} \tag{24}$$

$$f_3(x_4, x_5, x_6, c_3) \triangleq \mathbb{1}\{x_4 + x_5 + x_6 = c_3\} \tag{25}$$

と表記する．

このように生成された $\boldsymbol{c}$ を受け取り，復号器では式 (16) で表される条件付確率に従って復号を行う．ここで，この条件付確率の分母

$$\sum_{\boldsymbol{x}} P\{X^6 = \boldsymbol{x}\}\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\} \tag{26}$$

は $A\boldsymbol{x} = \boldsymbol{c}$ を満たす $\boldsymbol{x}$ が情報源から生成される確率の合計である．そのため，式 (26) は，符号語 $\boldsymbol{c}$ だけに依存する値となる．よって，符号語 $\boldsymbol{c}$ に対して式 (16) は定数 K を用いて

$$P\{X^6 = \boldsymbol{x}|C^3 = \boldsymbol{c}\} = \frac{1}{K} \times P\{X^6 = \boldsymbol{x}\}\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\} \tag{27}$$

と表すことができる．以上より，式 (16) を計算する際には，式 (27) の右辺に現われる $P\{X^6 = \boldsymbol{x}\}\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\}$ の値を求めれば良い．

しかし，$P\{X^6 = \boldsymbol{x}\}\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\}$ の計算を行う場合，その計算量は情報源系列 $\boldsymbol{x}$ の長さに対して，指数的に増加するため，実際に行うのは困難である．そのため，計算量を削減するために，シンボル単位で計算を行う方法が知られている．

例えば，符号語 $\boldsymbol{c}$ に対してシンボル x_1 に着目して計算すると，その事後確率 $P\{X_1 = x_1|C^3 = \boldsymbol{c}\}$ は，

$$P\{X_1 = x_1|C^3 = \boldsymbol{c}\} \tag{28}$$

$$= \sum_{x_2,x_3,x_4,x_5,x_6} P\{X^6 = x_1, \cdots, x_6|C^3 = \boldsymbol{c}\} \tag{29}$$

$$= \frac{1}{K} \times \sum_{x_2,x_3,x_4,x_5,x_6} P\{X^6 = x_1, \cdots, x_6\}\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\} \tag{30}$$

$$= \frac{1}{K} \times \sum_{x_2,x_3,x_4,x_5,x_6} f_1(x_1, x_2, x_3, c_1)f_2(x_3, x_4, c_2)f_3(x_4, x_5, x_6, c_3)\prod_{i=1}^{6} P\{X_i = x_i\} \tag{31}$$

となる．このような確率は事後確率を周辺化して得られるため，周辺事後確率と呼ばれている．

ここで，K は与えられた符号語 $\boldsymbol{c}$ に対して定数となる．このためシンボル x_1 の事後確率分布は，式 (31) から K を除いた

$$\sum_{x_2,x_3,x_4,x_5,x_6} f_1(x_1, x_2, x_3, c_1)f_2(x_3, x_4, c_2)f_3(x_4, x_5, x_6, c_3)\prod_{i=1}^{6} P\{X_i = x_i\} \tag{32}$$

の値によって決まる．この式 (32) はさらに，

$$\begin{aligned}
&\sum_{x_2,x_3,x_4,x_5,x_6} f_1(x_1,x_2,x_3,c_1)f_2(x_3,x_4,c_2)f_3(x_4,x_5,x_6,c_3)\prod_{i=1}^{6}P\{X_i=x_i\}\\
&=P_X(x_1)\\
&\times\sum_{x_2,x_3} f_1(x_1,x_2,x_3,c_1)P_X(x_2)P_X(x_3)\\
&\times\sum_{x_4} f_2(x_3,x_4,c_2)P_X(x_4)\\
&\times\sum_{x_5,x_6} f_3(x_4,x_5,x_6,c_3)P_X(x_5)P_X(x_6) \qquad (33)
\end{aligned}$$

と書き換えることができる．ここで，式 (33) の右辺にある $P_X(x_1)$ に着目すると，式 (33) の左辺では $P_X(x_1)$ が $x_2,\cdots,x_6$ の総和内に存在することで，それぞれの和が $P_X(x_1)$ と積を取る必要があった．これに対して，右辺では総和の外側へと $P_X(x_1)$ を出すことができ，一度の乗算で計算を終えることができる．このように，分配則を用いて，積と和で表されている式を適切な形にすることによって計算量を削減することができる．

3.2 ファクターグラフ

さらに，積計算を $M_{x_k\to f_i}(x_k)$，和計算を $M_{f_i\to x_k}(x_k)$ と置くことにより，式 (33) の計算過程を

$$\begin{aligned}
M_{x_1}(x_1) &\triangleq P_X(x_1)M_{f_1\to x_1}(x_1) && (34)\\
M_{f_1\to x_1}(x_1) &\triangleq \sum_{x_2,x_3} f_1(x_1,x_2,x_3,c_1)M_{x_2\to f_1}(x_2)M_{x_3\to f_1}(x_3) && (35)\\
M_{x_2\to f_1}(x_2) &\triangleq P_X(x_2) && (36)\\
M_{x_3\to f_1}(x_3) &\triangleq P_X(x_3)M_{f_2\to x_3}(x_3) && (37)\\
M_{f_2\to x_3}(x_3) &\triangleq \sum_{x_4} f_2(x_3,x_4,c_2)M_{x_4\to f_2}(x_4) && (38)\\
M_{x_4\to f_2}(x_4) &\triangleq P_X(x_4)M_{f_3\to x_4}(x_4) && (39)\\
M_{f_3\to x_4}(x_4) &\triangleq \sum_{x_5,x_6} f_3(x_4,x_5,x_6,c_3)M_{x_5\to f_3}(x_5)M_{x_6\to f_3}(x_6) && (40)\\
M_{x_5\to f_3}(x_5) &\triangleq P_X(x_5) && (41)\\
M_{x_6\to f_3}(x_6) &\triangleq P_X(x_6) && (42)\\
& && (43)
\end{aligned}$$

のように分解する．すると，式 (33) は図 7 のようなツリー構造を持つグラフで表すことができる．

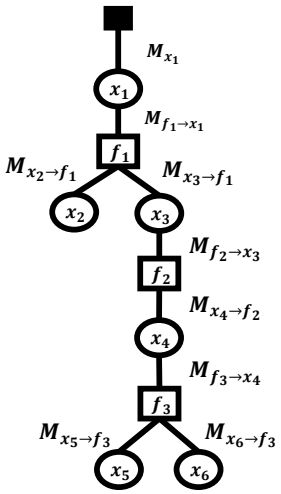

図 7　x_1 を根とするツリー

この図 7 は式 (33) の計算過程を視覚的に表したものとなる．各シンボル x_k と，シンボルが満たす制約条件 f_i をノードとし，積計算 $M_{x_k \to f_i}(x_k)$ と和計算 $M_{f_i \to x_k}(x_k)$ をエッジで表すことによって，計算過程をグラフ化している．このとき各シンボル x_k と，シンボルが満たす制約条件 f_i を表すノードをそれぞれ，変数ノード，関数ノードと呼ぶ．図 7 において，シンボル x_1 に対する周辺事後確率の計算は，図 7 のツリー下部から上部へと計算を持ち上げていくことに相当する．

さらに，$x_2, \cdots, x_6$ に対しても，同様の操作を行うことによって，図 7 と同じようにツリー構造を描くことが可能となる．例えばシンボル x_3 に対するグラフは図 8 のようになる．

ここで，図 7 と図 8 を比較すると，それぞれのグラフでノードとエッジの位置関係が等しく，グラフ全体の形状が同一であることが分かる．これは，異なるシンボルであっても符号を構成する行列は同一のものであり，式 (23) から (25) で表される制約条件と，各シンボルとの関係性も同一となることにより，$M_{x \to f}, M_{f \to x}$ で表されるエッジとノードとの位置関係が等しくなるためである．これは，その他のシンボルに対しても同様に言えることである．

そして，この性質を用いることで，全てのシンボルに対する周辺事後確率の計算を図 9 で表すように効率化できる．

図 9 は変数ノードを上に，関数ノードを下に配置することで上記のグラフを書き直したものであり，ファクターグラフと呼ばれる [4]．図 9 では，積和計算 $M_{x_k \to f_i}(x_k)$ と $M_{f_i \to x_k}(x_k)$ を，各シンボルに対して行うのではなく，全てのシンボルで同時に行うことで，計算量を削減している．これは，図 9 において，上のノードから下のノードへの計算 $M_{x_k \to f_i}(x_k)$ と，その

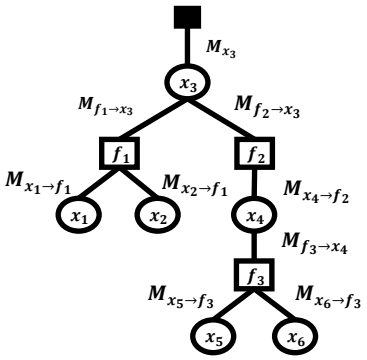

図 8　x_3 を根とするツリー

逆である $M_{f_i\to x_k}(x_k)$ を，繰り返し行うという意味である．

そしてこの $M_{x_k\to f_i}(x_k)$ と，$M_{f_i\to x_k}(x_k)$ の繰り返しには，繰り返しの必要回数が存在する．図 9 においてシンボル x_1 に着目すると，x_1 に対する周辺事後確率の計算には図 7 からも分かるように，x_5，x_6 からの情報伝達が必要となる．ここで，$M_{x_k\to f_i}(x_k)$ と $M_{f_i\to x_k}(x_k)$ の計算を一度ずつ行うことを，1 ステップとすると，仮に繰り返しが 1 ステップしか行われなかった場合，周辺事後確率の計算に必要な，シンボル x_5，x_6 の情報はシンボル x_4 までしか届かない．そのため，繰り返しが 1 ステップしか行われなかった場合，シンボル x_1 に対する周辺事後確率の計算を正しく行うことができなくなってしまう．この例では，シンボル x_5，x_6 の情報を伝達し，周辺事後確率の計算を行うためには，3 ステップが必要となる．このように，周辺事後確率を計算するための，$M_{x_k\to f_i}(x_k)$ と，$M_{f_i\to x_k}(x_k)$ の繰り返しには，必要回数が存在する．逆に考えると，周辺事後確率の計算には，この必要回数分計算を行えば十分であるということができる．よって，ファクターグラフを用いた上記の計算の終了条件とは，この必要回数分，繰り返し計算が行われることであると考えることができる．

上記のような，ファクターグラフ上でのメッセージ交換処理に基づくアルゴリズムは，Sum-Product アルゴリズムと呼ばれており [4]，本研究でもこのアルゴリズムを採用した．次節では，本節で具体例を用いて述べた Sum-Product アルゴリズムを一般化する．

ここで，この Sum-Product アルゴリズムが正しく機能するためには，図 7 で表されるような，ツリー下部から上部へと持ち上げる直線的なメッセージ交換が必要となる．そのため，グラフがこのメッセージ交換の流れを阻害するような構造を持つ場合，Sum-Product アルゴリズムが正しく計算を行う保証はない．そのような構造を持つグラフについては，第 4 章で詳細

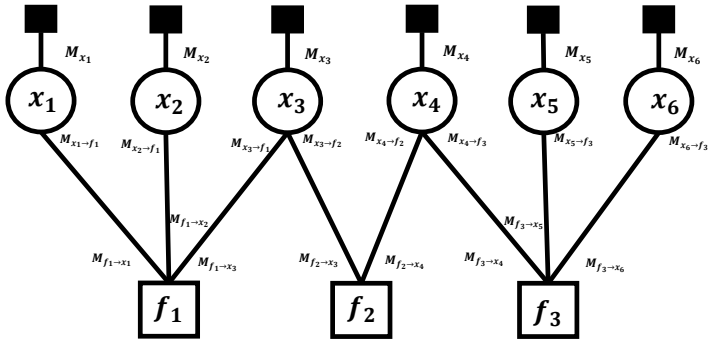

図 9　ファクターグラフ

に説明するが，次節では，そのような構造を持たない条件の下で Sum-Product アルゴリズムの一般化を行っている．

3.3　Sum-Product アルゴリズムの一般化

本節では，前節で具体例を用いて説明した Sum-Product アルゴリズムの処理手順を，文献 [4] を元に一般化する．

定義

符号化に使用する m 行 n 列の行列を A とし，変数集合を $V = \{x_1, \cdots, x_n\}$ とする．このとき，$A_1, \cdots, A_m$ を変数集合 V の部分集合とする．この部分集合 A_i の各要素は，行列 A の i 行目において，1 が立っている列の番号に対応する変数となる．例えば式 (18) の例では $A_1 = \{x_1, x_2, x_3\}$ となる．また，受信した符号語を $\boldsymbol{c} = (c_1, \cdots, c_m)$ とする．

このような行列 A のファクターグラフ G は変数ノードと関数ノードで構成されており，あるノード v に隣接するノードの集合を $N(v)$ と表記する．また，アルゴリズムの試行回数を変数 l で表し，初期値として $l = 1$ に設定する．そして，前節で述べたアルゴリズムの終了に必要な，反復回数を l_{max} と表す．

ステップ 1

各ノードごとでの処理を行う．変数ノードは関数ノードへメッセージを伝達し，関数ノードと変数ノードへメッセージを伝達する．以下にその処理手順をそれぞれ示す．

変数ノード処理

変数ノード x_k から関数ノード f_i へのメッセージを

$$M_{x_k \to f_i}(x_k) = \prod_{\alpha \in N(x_k) \setminus f_i} M_{\alpha \to x_k}(x_k) \tag{44}$$

として計算する．ただし，x_k が葉ノードの場合は，$M_{x_k \to f_i}(x_k) = 1$ とする (初期条件)．

関数ノード処理

関数ノード f_iから 変数ノード x_k へのメッセージを

$$M_{f_i \to x_k}(x_k) = \sum_{N(f_i) \setminus x_k} f_i(A_i,\ c_i) \prod_{\alpha \in N(f_i) \setminus x_k} M_{\alpha \to f_i}(\alpha) \tag{45}$$

として計算する．ただし f_i が葉ノードの場合，$M_{f_i \to x_k}(x_k) = f_i(x_k)$ とする（初期条件)．

ステップ 2

各ノードに伝達されたメッセージの積を計算することで，周辺化関数を導出する．

周辺化関数の計算

変数ノード x_k に対して，

$$M_{x_k}(x_k) = \prod_{\alpha \in N(x_k)} M_{\alpha \to x_k}(x_k) \tag{46}$$

を計算する．

ステップ 3

もし，$l < l_{max}$ ならば，$l = l + 1$ として，ステップ 1 に戻り同様の計算を行う．$l = l_{max}$ ならば計算を終了する．

Sum-Product アルゴリズムで正しく計算を行うためには，ステップ 2 で述べた，ノードとエッジ間のメッセージ交換が正確に行われる必要がある．ここで図 9 で表したように，ノードとエッジの関係性は使用する行列の形に深く影響する．そのため，次章では Sum-Product アルゴリズムが正しく機能するような行列の生成方法について述べる．

4　行列の生成方法

この章では，Sum-Product アルゴリズムの正確な計算に必要な行列の性質について述べる．そして，そのような行列の生成方法について，本研究で実装したものについて説明する．

4.1　タナーグラフ

まず，この節では行列とグラフとの関係性を述べる．ある行列が与えられたとき，その行列は 2 部グラフで表すことができる．ここで 2 部グラフとは，ノード集合が互いに排反な二つの集合に分割されており，エッジがそれら二つの集合間にのみ存在するグラフのことである．

このようなグラフは，タナーグラフと呼ばれ，文献 [4] で以下のように定義されている．

定義 2

2 元 $m \times n$ 行列 H の i 行 j 列成分を h_{ij} と書く．この行列を元に 2 部グラフ $G(H) = (V, E)$ を次のように定める．ノード集合 $V = V_1 \cup V_2$ の要素を

$$V_1 \triangleq \{v_1, v_2, \cdots, v_n\}, V_2 \triangleq \{c_1, c_2, \cdots, c_m\} \tag{47}$$

と名付ける．ここで，V_1 に含まれるノードを変数ノード，V_2 に含まれるノードをチェックノードと呼ぶ．このとき，$h_{ij} = 1 (i \in [1, m], j \in [1, n])$ を満たす (i, j) について，v_i と c_j をつなぐエッジ $e_{ij} \in E$ が存在する．また，$h_{ij} = 0 (i \in [1, m], j \in [1, n])$ ならば，v_i と c_j をつなぐエッジは存在しない．このように定義される 2 部グラフ $G(H)$ を H のタナーグラフと呼ぶ．

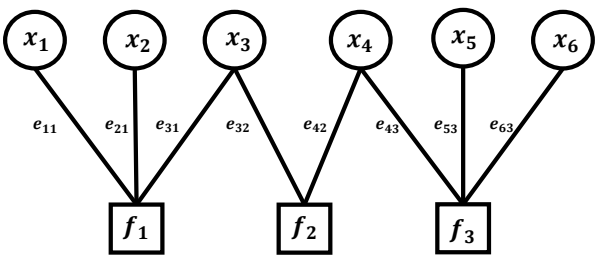

図 10　行列 A のタナーグラフ

上記の定義に基づく方法で，行列からタナーグラフを生成することができる．逆に定義 2 を満たすタナーグラフが与えられた際，そのタナーグラフから行列を生成することも可能である．すなわち，行列を生成しようとする際には，条件を満たすタナーグラフを生成すればよい．

ここで，前章の式 (18) で定義した行列 A について，定義 2 に従ってタナーグラフを生成すると図 10 のようになる．

このとき，図 10 で示したタナーグラフと，図 9 で示したファクターグラフを比較すると，ファクターグラフはタナーグラフを部分グラフに持つことが分かる．Sum-Product アルゴリズムはファクターグラフに基づいて計算が進められるため，部分グラフであるタナーグラフがどのようなものであるか考えることは，Sum-Product アルゴリズムを考える上で非常に重要である．

4.2 グラフの形と行列

前節では，タナーグラフについて述べた．ここで，以下で定義される行列 B について，そのタナーグラフを図 11 として描く．

$$B \triangleq \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{48}$$

ここで，図 11 のタナーグラフを，ノードとエッジの関係性を変えずに，その位置関係を並べ替えたグラフを図 12 として描く．

このとき，図 12 のタナーグラフの形状は，前節の図 10 で表されるタナーグラフの形状と等しいことが分かる．前節で述べたように，タナーグラフの形状は，Sum-Product アルゴリズムにおけるメッセージ交換則のもととなっている．そのため，図 12 と図 10 のように，行列が

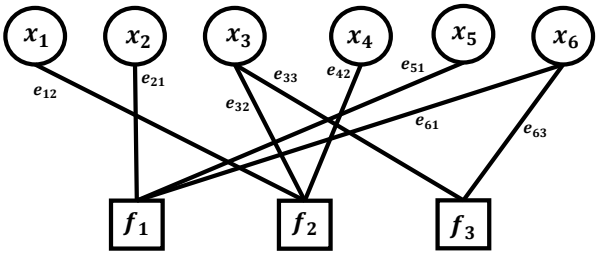

図 11　行列 B のタナーグラフ

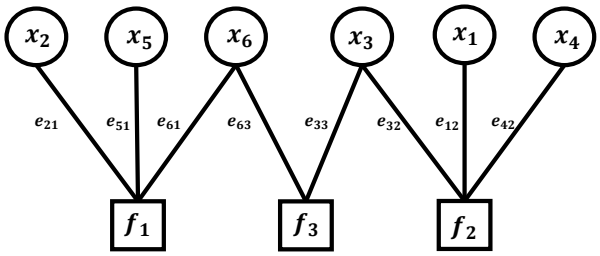

図 12　行列 B のタナーグラフの並べ替え

異なる場合でも，グラフの形状が等しいとき，Sum-Product アルゴリズムがたどる計算過程も等しくなる．よって，図 12 で表される行列 B と，図 10 で表される行列 A は，等しい性能を持つと考えることができる．

本研究では，グラフの形状を「タイプ」と呼称することにする．上記より，行列を生成するときに，タイプが等しければ，行列が持つ性能も等しい．そのため，行列を生成し，その性能差を調べたいときには，タイプが異なるもの同士を比較すれば十分である．次節以降では，タナーグラフと，そのタイプに焦点を当て，行列生成方法を説明する．

4.3　全域木

前節では，行列とタナーグラフ，タナーグラフとファクターグラフの関係性を述べた．本節では，第 3 章で触れた，Sum-Product アルゴリズムのメッセージ交換を阻害するようなタナーグラフの形状について述べる．

以下で定義される行列 C について，そのタナーグラフを図 13 として描く．

$$C \triangleq \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{49}$$

このとき，図 13 上のノード x_1 に着目すると，

$$x_1 \to e_{11} \to f_1 \to e_{12} \to x_2 \to e_{22} \to f_2 \to e_{21} \to x_1 \tag{50}$$

と経路をたどることで x_1 へと戻ることができる．このように，あるノードから出発し同じノードへと戻ってくるようなループ構造は，グラフ理論において「閉路」と呼ばれている [8]．Sum-Product アルゴリズムは，図 7 で表されるような，直線的なメッセージ交換の流れがも

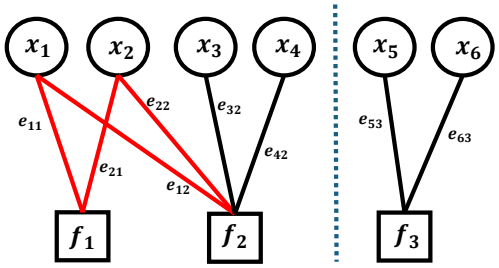

図 13　行列 C のタナーグラフ

とになっている．そのため，グラフに閉路が含まれる場合，Sum-Product アルゴリズムで正確な計算がなされる保証はなくなってしまう [4].

また，グラフ上の任意のノード二つがエッジで接続されていることは「連結」と呼ばれるが [8], 図 13 上のノード x_5, x_6, f_3 はそれ以外のノードと連結でない．前述の通り，Sum-Product アルゴリズムでは，各ノード間でのメッセージ交換をもとに計算を行う．そのため，グラフが連結でない場合，このメッセージ交換の流れが分断されることで，複数の独立した検査行列に対して，並列に計算しているのと同等になる．

上記の通り，Sum-Product アルゴリズムの性能を保証するようなタナーグラフは，閉路を持たないことが求められる．また，次節で述べるように，検証では連結なグラフを調べれば十分である．このような，閉路を持たず連結であるグラフは，グラフ理論において全域木 [8] と呼ばれており，上記の内容をまとめると，タナーグラフが全域木となるような行列を使用したとき，Sum-Product アルゴリズムの性能を保証することができる．

このように本節では，タナーグラフが全域木となるような行列を用いた場合，Sum-Product アルゴリズムで正しく計算できることを述べた．このような背景を踏まえて，次節では，この全域木と，実際に検証で使用する行列との関係性について説明する．

4.4　全域木と行列

前節では，タナーグラフが全域木となる行列が与えられたとき，Sum-Product アルゴリズムを用いて正しく計算ができることを述べた．しかし，実際の符号では，タナーグラフが全域木でない行列も使用することができる．本節では，符号に用いる行列と，全域木との関係性を述べ，実際に検証で使用する行列の考え方について説明する．

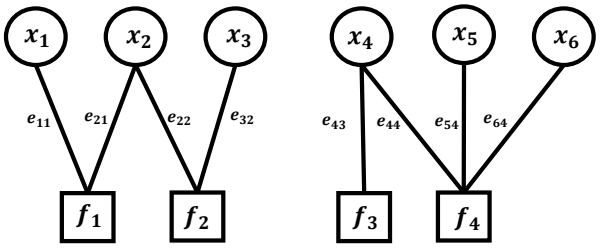

図 14　行列 D のタナーグラフ

6×3 行列 D を

$$D \triangleq \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{51}$$

のように定義すると，行列 D のタナーグラフは図 14 となる．図 14 からも分かるとおり，この行列のタナーグラフは連結でないため全域木とはならない．

ここで，この行列 D を用いて情報源系列 $\boldsymbol{x} = (x_1, \cdots, x_6)$ の符号化を行う場合，その符号語 $\boldsymbol{c} = (c_1, \cdots, c_4)$ は，行列 D と $\boldsymbol{x}$ の積であるため

$$c_1 = x_1 + x_2 \tag{52}$$
$$c_2 = x_2 + x_3 \tag{53}$$
$$c_3 = x_4 \tag{54}$$
$$c_4 = x_4 + x_5 + x_6 \tag{55}$$

と計算される．このとき，行列 D の 0 成分に注意すると，行列 D を用いて式 (52) から式 (55) で表される符号語 $\boldsymbol{c}$ を生成することは，行列 D を分割して得られる，2 つの行列 D_1 と D_2 を用いて，系列 $\boldsymbol{x_1} = (x_1, \cdots, x_3)$ と $\boldsymbol{x_2} = (x_4, \cdots, x_6)$ を

$$D_1 \boldsymbol{x_1} = \boldsymbol{c_1} \Leftrightarrow \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \tag{56}$$

$$D_2 \boldsymbol{x_2} = \boldsymbol{c_2} \Leftrightarrow \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} c_4 \\ c_5 \\ c_6 \end{pmatrix} \tag{57}$$

のように計算することと等しい．

ここで，図 14 より行列 D を分割して得られた，行列 D_1，D_2 のタナーグラフは全域木である．このため，もとの行列 D に対して Sum-Product アルゴリズムを適用するのではなく，分割したそれぞれの行列に適用すれば，タナーグラフが全域木がとならない，このような行列 D に対してでも，Sum-Product アルゴリズムを使用することができる．

上記のように，元の行列が全域木でなくても，その行列をタナーグラフが全域木となるような小さな行列に分割することができれば，Sum-Product アルゴリズムが適用できる．このことから，符号が持つ性能を考える上では，タナーグラフが全域木となる行列の性能に着目することが重要であることがわかる．このため，本研究ではタナーグラフが全域木となるような行列に焦点をあて，そのような行列を用いた性能検証を行った．

次節ではこれまで述べてきた，タナーグラフが全域木となるような行列の生成方法について詳細に述べる．

4.5 全域木の作成方法

本節では全域木の作成方法について，具体例を用いてその手順を説明する．この手法では行列のサイズが指定されたとき，その行列で考えられる，タイプが異なるタナーグラフを全て列挙する．また，作成されるタナーグラフは，前節で述べた全域木の条件を満たすものである．

図 15 は本研究で実装した，グラフ作成の流れを図示したものである．この例では，4×3 行列について，タイプの異なるタナーグラフを 7 つ列挙している．

作成方法について詳しく述べる．手順全体では，完成までの過程を階層化し，タナーグラフを 0 から生成している．各ステップでは前のステップで作成した全てのグラフに対して，変数ノードまたは関数ノードを一つずつ追加する．そして，追加したノードと既存のノードとをエッジでつなぐことによって，新たなグラフを作成する．この際，追加したノードと既存のノードのつなぎ方によって複数のグラフを作成可能であるため，その全てを列挙する．そして，このような，ノードとエッジの追加を繰り返すことによって，最終的に所望のノードとエッジを持つグラフを作成することができる．

ここで図 15 にあるように，各ステップでグラフを列挙する際，元となるグラフが異なる場合でも，追加するノードとエッジによっては，同じグラフが作成されてしまうことがある．このため，各ステップでグラフを列挙したあと，すぐに次のステップに移るのではなく，列挙したグラフの中で重複しているものがないか調べ，重複があった場合削除してから，次のステップに進むことによって，効率的に行列の生成を行っている．

さらに，このような手順でグラフを構成をすることによって，前節で述べた閉路を持たず連結であるグラフを作成することが可能である．まず，各手順で追加したノードは必ず既存のノードと接続される．そのため，全てのノードは連結となる．

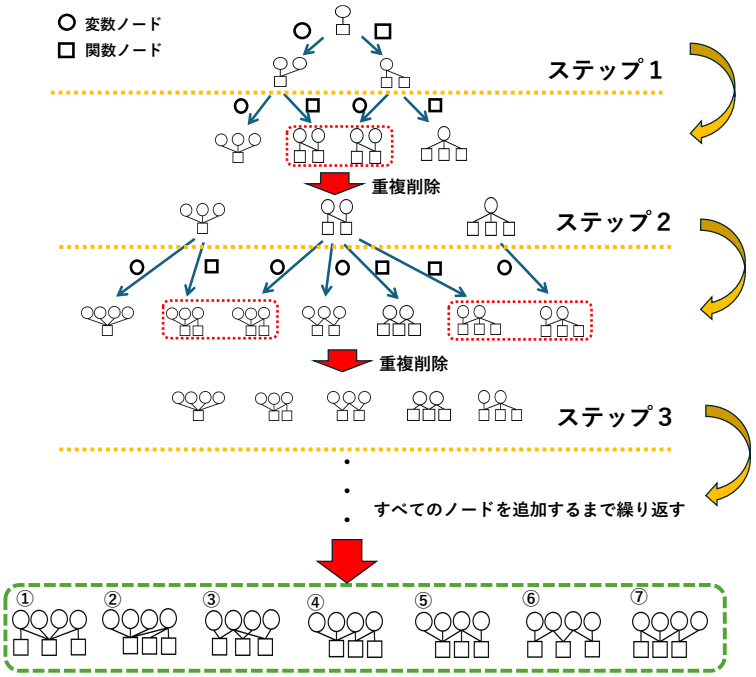

図 15　行列生成の手順

また，ノードが n 個のグラフにおいて，全てのノードが連結でありかつ，閉路を持つために必要なエッジの本数は，最低で n 本である．ここで，この手法ではノードを追加するたびに，新たに追加するエッジは 1 本となる．そのため，最終的に作成されるは n 個のノードを持つグラフにおいて，エッジは $n-1$ 本となる．よって，閉路を作るためのエッジの本数が足りないため，上記の手順で閉路を作ることは不可能となる．

本研究では上記の手法を用いて，行列生成を行っている．この手法を用いることで，行列のサイズを固定した上で，そのサイズで条件を満たした全てのタイプのタナーグラフを作ることができる．そのため，本研究では全てのタイプに対して性能を検証した．次章では，検証を行う上でその考え方のもととなる，数値計算方法について説明する．

5　数値計算による復号誤り確率の導出

本研究では，制約付き乱数による情報源符号化について，その性能を検証した．この際，検証では実際に送受信シミュレーションを行うのではなく，数値計算によって復号誤り確率を導

出することで，理論的な側面から性能検証を行った．本章では，制約付き乱数による情報源符号化と，その比較対象として実装したシンボル MAP 復号法 [4] について，それらの数値計算がどのようになされるか説明する．

5.1 シンボル MAP 復号の数値計算

シンボル MAP 復号法とは，シンボル単位で MAP(Maximum a Posteiriori Probability, 最大事後確率) を求める復号法である．具体的には，式 (16) における，シンボル x_i の周辺事後確率

$$P\{X_i = x_i | C^n = \boldsymbol{c}\} \tag{58}$$

に対して，シンボル MAP 復号では，その推定シンボル $\hat{x_i}$ が

$$\hat{x_i} = \underset{x_i \in \{0,1\}}{\arg\max}\ P\{x_i | C^n = \boldsymbol{c}\} \tag{59}$$

のように決定される．ここで，$\underset{x}{\arg\max}\ f(x)$ は $f(x)$ の最大値を与える x の値を意味する．すなわち，シンボル MAP 復号法では周辺事後確率を最大化する x_i が推定シンボルとして出力される．

上記のように，シンボル MAP 復号では，それぞれのシンボルに対する周辺事後確率を最大化するものを，推定シンボル $\hat{x_i}$ として出力する．そして，それらの推定シンボルを合わせて推定語 $\hat{\boldsymbol{x}}$ が求まる．このため，各シンボルに対する周辺事後確率が計算された段階で，推定語 $\hat{\boldsymbol{x}}$ は一つに定まる．

ここで，情報源符号化問題では，通信路での誤りが発生しないため，送りたい情報源系列 $\boldsymbol{x}$ によって決定される符号語 $\boldsymbol{c}$ は，誤りなく復号器へと届く．そして，周辺事後確率は復号器で受信した符号語 $\boldsymbol{c}$ によって決まる．よって，送信したい系列 $\boldsymbol{x}$ を決定した段階で，各シンボルに対する周辺事後確率は決まり，結果として推定語 $\hat{\boldsymbol{x}}$ も決まる．復号誤りは，推定語 $\hat{\boldsymbol{x}}$ と送信語 $\boldsymbol{x}$ とが，$\hat{\boldsymbol{x}} \neq \boldsymbol{x}$ となることであるから，復号誤りが発生するかどうかは，送信する系列 $\boldsymbol{x}$ を決めた時点で，決定されてしまう．

以上のことから，本研究のシンボル MAP 復号法における復号誤り確率 $\mathrm{Error}_{\mathrm{MAP}}$ は

$$\mathrm{Error}_{\mathrm{MAP}}(A) = 1 - \sum_{\boldsymbol{x}} P_X(\boldsymbol{x}) \prod_{i=1} \mathbb{1}\{\ x_i = \underset{x'_i \in \{0,1\}}{\arg\max}\ P_{X_i|C'}(x'_i | A\boldsymbol{x})\} \tag{60}$$

と表すことができる．ここで式 (60) 内の，$P_X(\boldsymbol{x})$ は情報源から系列 $\boldsymbol{x}$ が送信される確率であり，$\boldsymbol{x}$ による総和部分は復号「成功」確率を表している．

5.2 制約付き乱数による情報源符号化の数値計算

次に制約付き乱数による情報源符号化の復号誤り確率についても，その考え方を述べる．前節で示したとおり，この復号方法でも，送信したい系列 $\boldsymbol{x}$ を決定した段階で，各シンボルに対する周辺事後確率が定まる．しかし，シンボル MAP 復号法と異なり，制約付き乱数による復号では周辺事後確率の分布に従って，ランダムに各シンボルを決定し，さらに生成した推定語 $\hat{\boldsymbol{x}}$ が，制約条件 $A\hat{\boldsymbol{x}} = \boldsymbol{c}$ を満たすかどうか判別する．そして，制約条件を満たさなかった場合，満たすまで推定語の生成を繰り返す．

よって行列 A を使用したとき，この復号法における，復号誤り確率 $\mathrm{Error}_{\mathrm{CoCoNuTS}}$ は，情報源系列 $\boldsymbol{x}$ が送信される確率が $P_X(\boldsymbol{x})$ であることを考慮すると，

$$\mathrm{Error}_{\mathrm{CoCoNuTS}}(A) = \sum_{\substack{\boldsymbol{x},\hat{\boldsymbol{x}}:\\ \boldsymbol{x}\neq\hat{\boldsymbol{x}}}} P_X(\boldsymbol{x}) \times P_{X|C'}(\hat{\boldsymbol{x}}|A\boldsymbol{x}) \tag{61}$$

となる．式 (16) より，上記の式はさらに

$$\mathrm{Error}_{\mathrm{CoCoNuTS}}(A) = \sum_{\substack{\boldsymbol{x},\hat{\boldsymbol{x}}:\\ \boldsymbol{x}\neq\hat{\boldsymbol{x}}}} P_X(\boldsymbol{x}) \times \frac{P_X(\hat{\boldsymbol{x}})\mathbb{1}\{A\hat{\boldsymbol{x}} = A\boldsymbol{x}\}}{\sum_{\boldsymbol{a}} P_X(\boldsymbol{a})\mathbb{1}\{A\boldsymbol{a} = A\boldsymbol{x}\}} \tag{62}$$

と表すことができる．

このように式 (60) や式 (62) を計算することによって，実際に送信する系列をランダムに決定するようなシミュレーションを行うことなく，性能を検証することができる．本研究では，このようにして実装した符号を用いて検証を行ったため，次章では具体的な検証結果について述べる．

6 復号性能の検証

本研究では，制約付き乱数による情報源符号化について，その性能を検証した．行列生成を工夫することで，行列のサイズが等しく，それぞれのタイプが異なる行列を扱った．また，Sum-Product アルゴリズムを導入することで，様々な大きさの行列について検証を可能とした．

本章ではまず，同一サイズの行列における，それぞれのタイプが持つ性能とその分布について述べる．そして，制約付き乱数生成器が持つ性能を，比較対象であるシンボル MAP 復号法と比べ，考察を行う．

6.1　タイプの性能分布

図 16 は 10 × 7 の行列における，制約付き乱数を用いた復号法での，復号誤り確率とその分布を表したものである．10 × 7 行列では，本研究の行列生成アルゴリズムを用いることで，8710 のタイプを生成することができるため，その全てについて復号誤り確率を求めた．図 16 は横軸が復号誤り確率，縦軸が行列の個数の累積である．復号誤り確率は小さい方が良いため，グラフ左側に行くほど，そのタイプの行列が，良い性能を持つことになる．

図 16 から，復号誤り確率が小さくなるような性能の良い行列は，8710 タイプの中では，少ないことが分かる．

本研究では，行列のサイズを固定して，そのサイズで作成可能な全ての行列を作り，その中から性能の良いものを見つけ出している．しかし、行列のサイズが増大すると，作成可能な行列の個数も増加するため，その全てを列挙することは困難でである．仮に，生成された全ての行列の内，性能の良いもの割合が大きいのであれば，全数を列挙せずにランダムに行列生成をした場合でも，性能を保つことができる．しかし，図 16 の検証結果から分かる通り，性能の良いものが全体に占める割合は小さく，もし行列をランダムに生成した場合，性能が低くなってしまうと考えられる．

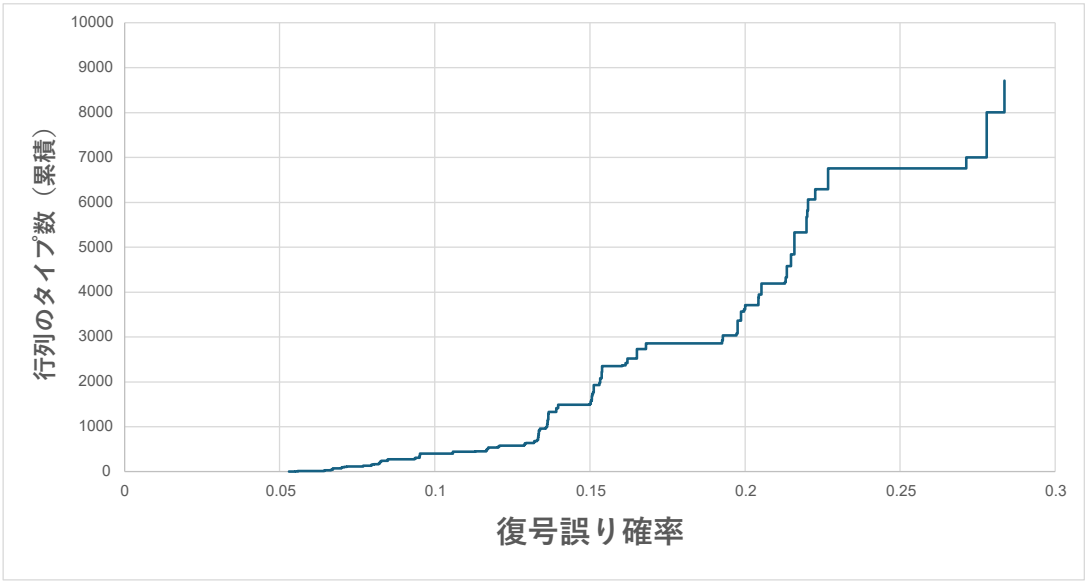

図 16　10 × 7 の行列の性能と分布

このことから，今後行列のサイズをさらに大きくしたい場合，行列生成については別の方法を考える必要がある．

6.2　複数の情報源における復号性能

図 17 は制約付き乱数を用いた符号の復号性能をまとめたグラフである．縦軸は復号誤り確率，横軸は情報源のエントロピーと符号化レートとの比である．ここで横軸について，式 (14) で表したように，誤りなく復元するためには符号化レート r は情報源のエントロピー $H(X)$ より大きくなくてはならない．そのため，検証に使用した行列の符号化レートは，情報源のエントロピーを越えるものであり，その際のレートと情報源のエントロピーとの比 $\frac{r}{H(X)}$ を横軸としている．

また，検証は同一サイズの行列ごとに行い，その中でタイプの異なる行列同士の復号誤り確率を比較している．このようにして求めた復号誤り確率の中で，最も小さいものを図 17 にプロットしている．そして，情報源の分布を $P_X(1) = \{0.75, 0.8, 0.85, 0.9\}$ と変えることで，情報源の異なる複数の条件で行った．このとき，図 17 において性能の良い符号とは，誤り確率が小さく，符号化レートが小さいものである．そのため，グラフの左下にいくほど性能が良いものとなる．

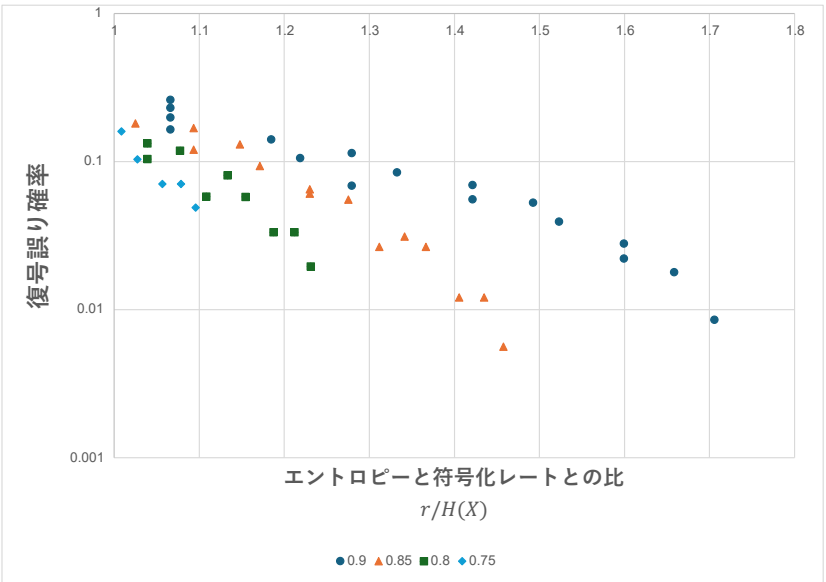

図 17　制約付き乱数を用いた符号の復号性能

図 17 からは 2 つのことが読み取れる．まず，全ての情報源においてグラフの右側へ行くほど，誤り確率が小さくなった．また，プロットされた点を情報源ごとに比較すると，$P_X(1)$ が小さくなるにつれて，点が左下へと近づいていく．

前者について，グラフの右側へ行くということは，情報源のエントロピーに対して，レートが大きいということであり，情報源符号化においては，情報をあまり圧縮していないということに相当する．そのため，圧縮率の高い左側の符号に比べて，容易に復元することができ，誤りが抑えられたのではないかと考えられる．

また，後者について，$P_X(1)$ が小さくなるということは，式 (3) で定義される，情報源のエントロピーが増大するということである．このとき，$r > H(X)$ を実現するためには，符号化レート r も同時に増大させる必要がある．よって，異なる情報源同士を比較した場合，横軸である $\frac{r}{H(X)}$ が同じような値でも，$P_X(1)$ が小さくなるほど，符号のレート r が大きい，つまり圧縮率の低い符号が使用されているため，復号誤り確率が小さくなり，結果として $P_X(1)$ が小さくなるほど，グラフ上の点が左下に近づくと考えられる．

次に同じデータを横軸の表現を変え，図 18 としてプロットした．図 17 は横軸を，情報源のエントロピーと符号化レートとの比で表したの対して，この図 18 ではダイバージェンスを横軸としている．ダイバージェンスとは，情報源のエントロピー $H(X)$ に対するレート r の冗

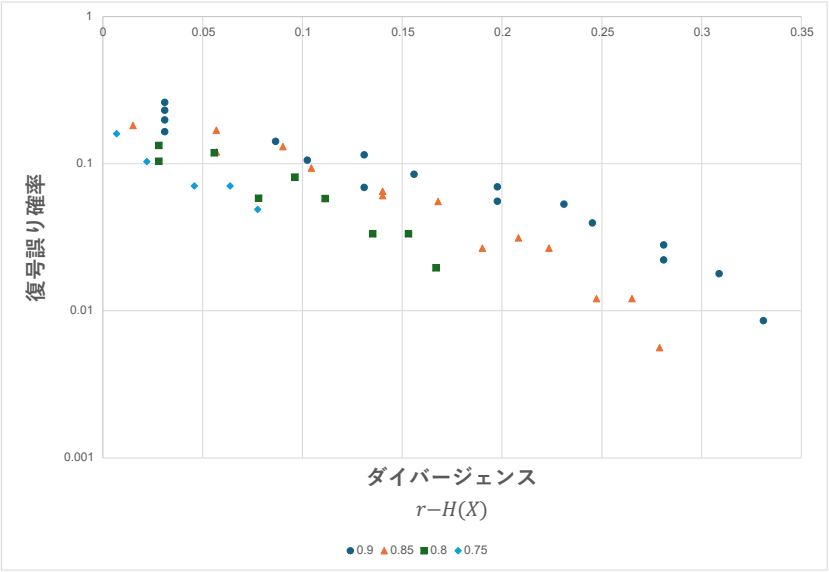

図 18　制約付き乱数を用いた符号の復号性能 (ダイバージェンス)

長性を表す尺度であり，$r - H(X)$ で表現される．

図 17 と図 18 を比較すると分かる通り，横軸を $\frac{r}{H(X)}$ とした場合と，ダイバージェンス $r - H(X)$ とした場合で，各点の位置関係に殆ど変化はない．そのため，ダイバージェンスを横軸とした図 18 に対して，上記の図 17 で行ったものと，同様の考察を行うことができる．

このように，制約付き乱数を用いた符号では，レート r を大きくすると復号誤り確率が小さくなる．乱数を用いた符号のこの性質は，情報源符号化定理で示されているものと一致している [5]．このため，制約付き乱数を用いた情報源符号化は，乱数を用いるという点で他の符号と異なるが，符号が持つ性質自体は一般の符号と同様に考えることができる．

さらに，図 17 と図 18 からは復号誤り確率が小さくなる様子についても考察を行うことができる．図 17 と図 18 では縦軸を復号誤り確率としたが，その目盛りは対数目盛を使用している．そのため，このグラフにおいて誤り確率の減少が直線的である場合，その減少度合いは指数的あると言うことが出来る．

ここで上記の通り，図 17 と図 18 では各情報源においてグラフの右側に行くほど，その復号誤り確率が小さくなるが，その際にプロットされる点は直線的ではなく，上に凸な曲線のような形でグラフ右下へと続いている．そのため，制約付き乱数による復号法では，横軸に対する復号誤り確率の減少度合いが，指数的なものよりも，さらに大きいものであると言える．

このように，図 17 と図 18 からは，制約付き乱数による復号法と情報源符号化定理との関係性や，復号誤り確率の減少度合いに関する特徴を読み取ることができた．次節では，他の復号方法と比較することで，制約付き乱数による復号法の性能についてより詳しく述べる．

6.3 制約付き乱数による復号法とシンボル MAP 復号法との比較

シンボル MAP 復号法との比較結果を図 19 に示す．グラフの縦軸，横軸の考え方は先ほどと同様である．図 19 は $P_X(1) = 0.9$ の場合の検証結果である．

シンボル MAP 復号法と制約付き乱数による復号法では，符号性能は等しいか，後者の方が優れていることが分かる．これは，制約付き乱数による符号の復号方法が関係していると考えることができる．

シンボル MAP 復号法では，各シンボルを式 (59) のように，周辺事後確率にしたがって，一度の操作で決定する．これに対して，制約付き乱数による復号法では各シンボルをランダムに決定した後に，制約条件 $A'\boldsymbol{x} = \boldsymbol{c'}$ 満たすか判別し，満たすまで生成を繰り返す．このため，生成される推定語は制約条件を満たしたもののみになり，結果としてシンボル MAP 復号法と比較して復号性能をが向上したのではないかと考えられる．

このように，制約条件を満たす推定語を生成することによって，制約付き乱数による復号法は高い性能を保っている．しかし，より大規模な通信を試みるために，行列のサイズを大きくした場合，送信される情報源系列も長くなり，結果として制約を満たすまで生成を行う操作

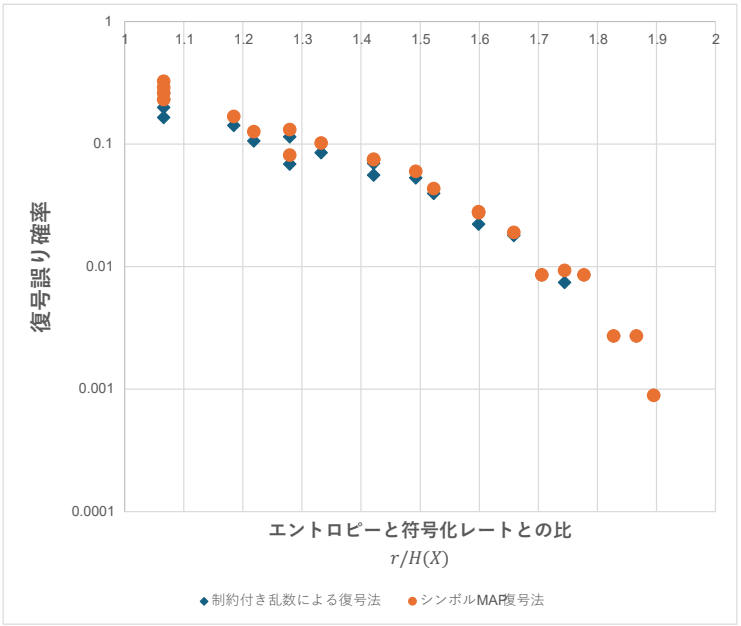

図 19　$P\{x=1\}=0.9$ の情報源における，制約付き乱数による復号法，シンボル MAP 復号法の比較

に，時間がかかってしまう．そのため，制約を満たす系列の生成方法を改良していくことが，行列を大きくしていく上での今後の課題となる．

7　まとめ

本研究では，制約付き乱数生成器による情報源符号化に着目し，その性能を検証した．また，符号で使用する行列の生成方法も工夫し，行列が持つ性能を効率的に探索することができた．制約付き乱数生成器を用いた符号では，通常の符号と比較して高い性能を持つことが確認できた．今後の課題としては，行列生成方法，情報源符号化における復号方法の両面で，行列サイズの増大に伴う改良の必要性が上げられる．また，第 2 章で述べた，補助情報付き情報源符号化や，CoCoNuTS 本来の考え方である，通信路符号化に対する検証も進めていく必要がある．

謝辞

本研究を行うにあたり，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.

参考文献

[1] C.E. Shannon, “A mathematical theory of communication,” Bell System Technical Journal, vol.27, pp.379–423, 623–656, 1948.

[2] Jun Muramatsu, Shigeki Miyake, “Channel Code Using Constrained-Random Number Generator Revisited,” IEEE Transactions on Information Theory, vol.65, no.1, pp.500-510, Jan. 2019.

[3] 南澤航,「制約付き乱数に基づく符号の Z 通信路に対する基礎的検証」，信州大学工学部卒業論文（指導教員：西新幹彦），2023 年 3 月.

[4] 和田山正，誤り訂正技術の基礎，森北出版，2010.

[5] T. Cover, J. Thomas, (山本博資，古賀弘樹，有村光晴，岩本貢訳)，情報理論　基礎と広がり，共立出版株式会社， 2012.

[6] P. Elias, “Coding for noisy channels,” IRE Conv. Rec, vol.3, pp.37–46, 1955.

[7] R.L. Dobrushin, “Asymptotic optimality of group and systematic codes for some channels,” Theory of Probability and Its Applications, vol.8, no.1, pp. 47–60, 1963.

[8] R. Ahlswede, “Group codes do not achieve Shanon’s channel capacity for general discrete chanenels,” The Annals of Mathematical Statistics, vol.42 , no.1, pp.224–240, 1971.

[9] R. Wilson, (西関隆夫，西関裕子訳)，グラフ理論入門，近代科学社， 2018.

付録 A　補助情報付き情報源符号化問題

A.1　復号器のみに補助情報が与えられる場合

復号器のみに補助情報が与えられる，補助情報付き情報源符号化問題について，その計算過程を説明する．通信のモデルは図 5 となる．このとき，復号器では，符号語 $\boldsymbol{c}$ と補助情報 $\boldsymbol{y}$ を受け取り，事後確率分布 $P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c})$ に従って，送信された系列 $\boldsymbol{x}$ を推定する．ここで，$P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c})$ は

$$P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c}) = \frac{P_{C|YX}(\boldsymbol{c}|\boldsymbol{y},\boldsymbol{x})P_{X|Y}(\boldsymbol{x}|\boldsymbol{y})}{P_{C|Y}(\boldsymbol{c}|\boldsymbol{y})} \tag{63}$$

$$= \frac{1}{P_{CY}(\boldsymbol{c},\boldsymbol{y})} \times P_{C|YX}(\boldsymbol{c}|\boldsymbol{y},\boldsymbol{x})P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})P_X(\boldsymbol{x}) \tag{64}$$

と表すことができる．このとき，式 (64) 内の式 $P_{C|YX}(\boldsymbol{c}|\boldsymbol{y},\boldsymbol{x})$ について，この式は $\boldsymbol{y},\boldsymbol{x}$ を受け取ったもとで $\boldsymbol{c}$ が何であるかを表しており，図 5 より符号語 $\boldsymbol{c}$ の決定には $\boldsymbol{y}$ は関与していない．また，$\boldsymbol{x}$ が決定すれば $\boldsymbol{c}$ も一つに定まるため，式 $P_{C|YX}(\boldsymbol{c}|\boldsymbol{y},\boldsymbol{x})$ は

$$P_{C|YX}(\boldsymbol{c}|\boldsymbol{y},\boldsymbol{x}) = \mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\} \tag{65}$$

となる．よって式 (64) は

$$P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c}) = \frac{1}{P_{CY}(\boldsymbol{c},\boldsymbol{y})} \times P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})P_X(\boldsymbol{x})\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\} \tag{66}$$

と書き直すことができる．

ここで，情報源 X と補助情報源 Y を定常無記憶情報源とすると，条件付き確率 $P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})$ は

$$P_{Y|X}(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1} P_{Y_i|X_i}(y_i|x_i) \tag{67}$$

と表すことができる．この式 (67) を用いると，式 (73) のシンボル x_i に対する周辺事後確率分布 $P_{X_i|YC}(x_i|\boldsymbol{y},\boldsymbol{c})$ は

$$P_{X_i|YC}(x_i|\boldsymbol{y},\boldsymbol{c}) = \sum_{\backslash x_i} P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c}) \tag{68}$$

$$= \sum_{\backslash x_i} \frac{1}{P_{CY}(\boldsymbol{c},\boldsymbol{y})} \times P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})P_X(\boldsymbol{x})\mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\} \tag{69}$$

$$= \frac{1}{P_{CY}(\boldsymbol{c},\boldsymbol{y})} \times \sum_{\backslash x_i} \mathbb{1}\{A\boldsymbol{x} = \boldsymbol{c}\} \prod_{i=1} P_{Y_i|X_i}(y_i|x_i)P_{X_i}(x_i) \tag{70}$$

となる．

行列を用いた実際の計算では，式 (70) の $\mathbb{1}\{A\boldsymbol{x}=\boldsymbol{c}\}$ は，第 3 章の式 (23) から式 (25) と同様に，関数 f を用いて分解することができる．よって，式 (70) の周辺事後確率は，第 3 章で述べた Sum-Product アルゴリズムを適用可能であり，その値を具体的に求めることができる．

A.2　符号器，復号器に補助情報が与えられる場合

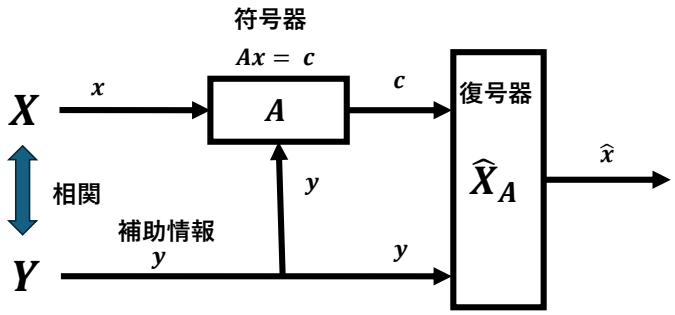

図 20　符号器，復号器で補助情報を共有する補助情報付き情報源符号化モデル

図 20 は補助情報付き情報源符号化問題の中でも，符号器，復号器の両方に補助情報を与える場合のモデルである．このとき復号器では，付録 A と同様に事後確率 $P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c})$ に従って，送信された系列 $\boldsymbol{x}$ を推定する．このため，事後確率 $P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c})$ は付録 A の式 (64) と同様に変形することができる．

ここで，図 20 の問題設定では，符号器で補助情報 $\boldsymbol{y}$ を受け取るため，符号器では情報源系列 $\boldsymbol{x}$ と補助情報 $\boldsymbol{y}$ を用いて，

$$A(\boldsymbol{x}+\boldsymbol{y})=\boldsymbol{c} \tag{71}$$

と計算することで，符号語 $\boldsymbol{c}$ を生成する．このため，式 (64) の $P_{C|YX}(\boldsymbol{c}|\boldsymbol{y},\boldsymbol{x})$ について付録 A の問題設定と異なり，計算には補助情報 $\boldsymbol{y}$ が必要となる．よって $P_{C|YX}(\boldsymbol{c}|\boldsymbol{y},\boldsymbol{x})$ は

$$P_{C|YX}(\boldsymbol{c}|\boldsymbol{y},\boldsymbol{x})=\mathbb{1}\{A(\boldsymbol{x}+\boldsymbol{y})=\boldsymbol{c}\} \tag{72}$$

と表される．上記の式 (72) より，事後確率 $P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c})$ は

$$P_{X|YC}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{c})=\frac{1}{P_{CY}(\boldsymbol{c},\boldsymbol{y})}\times P_{Y|X}(\boldsymbol{y}|\boldsymbol{x})P_X(\boldsymbol{x})\mathbb{1}\{A(\boldsymbol{x}+\boldsymbol{y})=\boldsymbol{c}\} \tag{73}$$

と表すことができ，シンボル x_i に対する周辺事後確率 $P_{X_i|YC}(x_i|\boldsymbol{y},\boldsymbol{c})$ は，付録 A と同様に考えることで，

$$P_{X_i|YC}(x_i|\boldsymbol{y},\boldsymbol{c}) = \frac{1}{P_{CY}(\boldsymbol{c},\boldsymbol{y})} \times \sum_{\backslash x_i} \mathbb{1}\{A(\boldsymbol{x}+\boldsymbol{y})=\boldsymbol{c}\} \prod_{i=1} P_{Y_i|X_i}(y_i|x_i) P_{X_i}(x_i) \qquad (74)$$

と表すことができる．

式 (74) の付録 A との相違点は，$\mathbb{1}\{A(\boldsymbol{x}+\boldsymbol{y})=\boldsymbol{c}\}$ の部分である．行列を用いた実際の計算では，$\mathbb{1}\{A(\boldsymbol{x}+\boldsymbol{y})=\boldsymbol{c}\}$ を，付録 A と同様に式 (23) から式 (25) で表されるような，関数 f を使った式に変形することができる．この際，補助情報 $\boldsymbol{y}$ は符号器と復号器で共有されているため，関数 f の中では定数として扱うことができる．

上記のように式を分解することで，付録 A と同様 Sum-Product アルゴリズムを適用可能であり，その値を具体的に求めることができる．

付録 B　ソースコード

B.1　全域木の生成を行うプログラム

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import networkx.algorithms.isomorphism as iso
import copy

ROW = 2
COLUMN = 3

all_edges = []

for i in range(COLUMN):       #頂点がすべてつながっている検査行列を作成
    for j in range (COLUMN, ROW + COLUMN, 1):
        all_edges.append((i,j))



#print(all_edges)

kouho_edges = []

kouho_edges.append([all_edges[0]])



count = 0

new_kouho_edges = []
new_kouho_edges.append([all_edges[0]])

graph_list = []


G = nx.Graph()
G.add_edge(all_edges[0][0], all_edges[0][1])
G.add_node(all_edges[0][0])
G.add_node(all_edges[0][1])
```

```
G.nodes[all_edges[0][0]]['parity'] = 'symbol'
G.nodes[all_edges[0][1]]['parity'] = 'f'
graph_list.append(G)


while count < ROW + COLUMN - 2:
    kouho_edges = new_kouho_edges
    print("count = ", count)
    new_kouho_edges = []
    tuika_suruyatu = []
    new_graph_list = []
    hash_list = []

    for n in range(len(kouho_edges)):
        eraban = []
        tuika_suruyatu.append([])

        for l in range(2):  #○を追加するか□を追加するか
            check = ROW + COLUMN  #あり得ない数字
            kouho_edges[n].sort(key = lambda x: x[l]) #小さい順に

            for i in range(len(kouho_edges[n])):
                eraban.append(kouho_edges[n][i][not l]) #すでに追加してあるものを抽出

            eraban_list = list(set(eraban))  #同一のものを削除

            for i in range(len(kouho_edges[n])):
                if check != kouho_edges[n][i][l]:  #同一のノードの追加無し
                    for j in range(len(all_edges)):
                        if kouho_edges[n][i][l] == all_edges[j][l]  and (all_edges[j][not l] in eraban_list) == False:
                            tounyu = copy.deepcopy(kouho_edges[n])
                            #print("tou", tounyu)
                            #print("all", all_edges[j])
                            tounyu.append(all_edges[j])
                            new_kouho_edges.append(tounyu)
                            #print(new_kouho_edges)
                            #print("koukooou", kouho_edges)
                            check = kouho_edges[n][i][l]
                            #tuika_suruyatu[n].append((all_edges[j][not l] ,all_edges[j]))

                            gura = copy.deepcopy(graph_list[n])
                            gura.add_edge(all_edges[j][0], all_edges[j][1])
                            gura.add_node(all_edges[j][not l])

                            if all_edges[j][not l] < COLUMN:
                                gura.nodes[all_edges[j][not l]]['parity'] = 'symbol'
                                #print("pow")

                            else:
                                gura.nodes[all_edges[j][not l]]['parity'] = 'f'
                                #print("piw")

                            new_graph_list.append(gura)
                            hash_list.append(nx.weisfeiler_lehman_graph_hash(gura, node_attr = "parity"))
                            break

                    nm = iso.categorical_node_match("parity", "f")

                    #print(len(new_graph_list))
                    if len(new_graph_list) > 1:
                        for j in reversed(range(0, len(new_graph_list) - 1, 1)):
                            #print("ooooooooooooooooo")
                            if (hash_list[-1] == hash_list[j]):
                                if (nx.is_isomorphic(new_graph_list[-1], new_graph_list[j], node_match = nm) == True):
                                    del new_graph_list[-1]
                                    del new_kouho_edges[-1]
                                    del hash_list[-1]
                                    break
```

```
    graph_list = copy.deepcopy(new_graph_list)
    count += 1

print(len(new_kouho_edges))


"""
for i in range(len(new_kouho_edges)):
    print(new_kouho_edges[i])
"""

del_list = []


for i in range(len(new_kouho_edges)):
    check_pari = np.zeros((ROW, COLUMN))
    for j in range(len(new_kouho_edges[i])):
        check_pari[new_kouho_edges[i][j][1] - COLUMN][new_kouho_edges[i][j][0]] = 1

    if ROW != np.linalg.matrix_rank(check_pari):
        del_list.append(i)

a = 0

for i in del_list:
    del new_kouho_edges[i - a]
    a += 1


f = open('Tree_4.txt', 'w')
f.write(str(COLUMN))
f.write("\t")
f.write(str(ROW))
f.write("\t")
f.write(str(len(new_kouho_edges)))
f.write("\n")
f.close()

f = open('Tree_4.txt', 'a')
for i in range(len(new_kouho_edges)):
    for j in range(len(new_kouho_edges[i])):
        for l in range(2):
            f.write(str(new_kouho_edges[i][j][l]))
            f.write("\t")
        f.write("\n")
f.close()

#print(tuika_suruyatu)
```

B.2 制約付き乱数に基づく情報源符号化を行うプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <time.h>
#include "MT.h"

#define BEST_G 100
#define WORST_G 100

double g[2];
double cc[2];
double ss[2];
```

```
double gg[2];

double s_p;             //情報源の分布（1 の出る確率）
int Hon;                //1 が立っている場所の数
int p_row, p_column;        //検査行列の行数、列数
int loop;  //サムプロダクト実施回数
int max;


double (*symbol_dis)[2];



typedef struct
{
    int column;
    int row;
    double rfx[2];
    double qxf[2];
    int best_column[3][BEST_G];
    int best_row[3][BEST_G];
    int worst_column[3][WORST_G];
    int worst_row[3][WORST_G];

} Par;

Par *lim;

double (*dis_pari)[3];

double ux(int x){
    if(x == 0){
        return (1 - s_p);
    } else {
        return (s_p);
    }
}







int encoder(int x_hat[], int c[]){
    int i, j;
    int a;
    int ten;

    a = 0;
    for (i = p_column; i < (p_row + p_column); i++){
        //printf("%d", a);
        c[a] = 0;
        for (j = 0; j < Hon; j++){
            //printf("%d  %d\n", lim[j][1], i);
            if (lim[j].row == i){
                //printf("wooo\n");
                c[a] ^= x_hat[lim[j].column];
                //printf("%d, %d\n",c[a], x_hat[lim[j][0]]);
            }
        }
        a++;
    }


    ten = 0;
    for (i = 0; i < p_row; i++){
        ten += c[i] << i;
    }
```

```
    //printf("tetet%d\n", ten);

    return ten;
}




void decoder(int c[]){
    int i, j, n;
    int loop_sum_product;
    int m, l;
    int length;
    int sum;
    unsigned int max1;  //32 ビットフルで使う
    double check_sum[2];
    int index[p_column];
    double normal_const;
    int likely_word;
    double max2;


    //検査行列の 1 がある部分に初期値を入れる
    for (i = 0; i < Hon; i++){
        for (l = 0; l < 2; l++){
            lim[i].rfx[l] = 0.0;
            lim[i].qxf[l] = 1.0 / 2.0;
            //printf("%f",qxf[l][i]);
        }
        //printf("%f", qxf[l][i]);
    }

    loop_sum_product = 0;

    while (loop_sum_product < loop){
        //printf("%d\n", loop_sum_product);
        for (i = 0; i < Hon; i++){
            //printf("%d\n",i);
            length = 0;
            for (l = 0; l < 2; l++){
                    check_sum[l] = 0.0;
            }
            for (j = 0; j < Hon; j++){
                if ((lim[j].row == lim[i].row) && (i != j)){
                    index[length] = j;
                    length++;
                }
            }

            if (length > 32){
                printf(" gyouretu ookisugi [%d]\n", __LINE__);
                break;
            }

            //printf("len%d\n", length);
            max1 = 1 << length;

            //printf("i ==%d  %d\n",i, 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++){
                        //printf("len = %d\n", length);
```

```
                    sum ^=  ((m >> n) & 1);
                    //printf("i == %d,  m == %d , sum == %d\n", i, ((m >> n) & 1), sum);
                }

                if ((sum ^ l) != c[lim[i].row - p_column]) continue;

                for (n = 0; n < length; n++){
                    ss[l]
                    *=
                    lim[index[n]].qxf[(m >> n) & 1] *
                    ux((m >> n) & 1);

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

                //printf("i ==%d  %f\n",i ,ss[l]);

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

        }



        normal_const = 0.0;
        for (l = 0; l < 2; l++){
            //printf("i == %d  %f\n", i, check_sum[l]);
            normal_const += check_sum[l];

        }

        //printf("i = %d  %f\n",i, normal_const);
        for (l = 0; l < 2; l++){
            lim[i].rfx[l] = check_sum[l] / normal_const;
            //printf("koou%d = %f\n",i, rfx[l][i]);
        }
    }



    for (i = 0; i < Hon; i++){
        for (l = 0; l < 2; l++){
            cc[l] = 1.0;
        }
        for (j = 0; j < Hon; j++){
            if ((lim[j].column == lim[i].column) && (i != j)){
                for (l = 0; l < 2; l++){
                    cc[l] *= lim[j].rfx[l];
                }
            }

        }

        normal_const = 0.0;

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

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

    }
    loop_sum_product++;
}
```

```
    for (i = 0; i < p_column; i++){
        for (l = 0; l < 2; l++){
            gg[l] = 1.0;
            for (j = 0; j < Hon; j++){
                if (lim[j].column == i){
                    gg[l] *= lim[j].rfx[l];
                }
            }
            gg[l] *= ux(l);
        }
        normal_const = 0.0;

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

        for (l = 0; l < 2; l++){
            symbol_dis[i][l] = 0;
            g[l] = gg[l] / normal_const;
            symbol_dis[i][l] = g[l];

        }

        //printf("P%d  0 = %f   1 = %f\n", i, symbol_dis[i][0], symbol_dis[i][1]);

    }
    return;
}



int count[3] = {0,0,0};
double cal[3] = {0,0,0};
//int sum_coco1_time = 0;

void error_count(int send){
    int i, j, k;
    int x_hat[p_column];
    int x_kamo[p_column];
    int x_check[p_column];
    int x[p_column];
    int c[p_row];
    int c_check[p_row];
    double syn_mul, syn_sum, syn_mul_send;
    double sou_mul, mapmax, sou_mul_map;
    int map_j, a, ten, tenten;

    sou_mul = 1;

    for (i = 0; i < p_column; i++){
        //printf("send = %d\n", send);
        x_hat[i] = (send >> i) & 1;



        sou_mul *= (x_hat[i] == 0) ? 1 - s_p : s_p;
    }




    //printf("xhat = ");
    for (i = 0; i < p_column; i++){
        //printf("%d", x_hat[i]);
    }
    //putchar('\n');
    //printf("////////////////////////////////////\n");
```

```
ten = encoder(x_hat, c);


decoder(c);


//map
sou_mul_map = sou_mul;
for (i = 0 ; i < p_column; i++){
    if (fabs(symbol_dis[i][0] - 0.5) <= 0.000000000001){
        x_kamo[i] = 3;

    } else if (symbol_dis[i][0] > symbol_dis[i][1]) {
        x_kamo[i] = 0;
    } else {
        x_kamo[i] = 1;
    }
}

for (i = 0; i < p_column; i++){
    if (x_kamo[i] == 3) {
        sou_mul_map = sou_mul_map / 2;
    }
    else if (x_kamo[i] != x_hat[i]) {
        sou_mul_map = 0;
        break;
    }
}

cal[0] += sou_mul_map;




for (j = 0; j < max; j++){
    syn_mul_send = 1;
    if (j == send){
        for (i = 0; i < p_column; i++){
            syn_mul_send *= symbol_dis[i][(j >> i) & 1];
            //printf("%f   ", symbol_dis[i][(j >> i) & 1]);
        }
        break;
    }
    //printf("%d  syn_smul == %f\n", j, syn_mul_send);
}
//putchar('\n');

cal[1] += sou_mul * syn_mul_send;

//printf("syn_sum == %f\n", syn_sum);



//制約あり CoCoNuTS
syn_sum = 0;
for (i = 0; i < max; i++){
    syn_mul = 1;
    tenten = 0;
    for (j = 0; j < p_column; j++){
        x_check[j] = (i >> j) & 1;
    }
    a = 0;
    for (j = p_column; j < (p_row + p_column); j++){
        //printf("%d", a);
        c_check[a] = 0;
        for (k = 0; k < Hon; k++){
            //printf("%d  %d\n", lim[j][1], i);
            if (lim[k].row == j){
                //printf("wooo\n");
```

```
                    c_check[a] ^= x_check[lim[k].column];
                    //printf("%d, %d\n",c[a], x_hat[lim[j][0]]);
                }
            }
            a++;
        }

        for (j = 0; j < p_row; j++){
            tenten += c_check[j] << j;
        }

        //printf("ten = %d   tenten = %d\n", ten, tenten);

        if (ten == tenten){
            for (j = 0; j < p_column; j++){
                syn_mul *= symbol_dis[j][x_check[j]];
                //printf("synmul ==  %f\n", syn_mul);
            }
            syn_sum += syn_mul;
        }
        //printf("ooooo ==  %f\n", syn_mul);


    }

    //printf("suuu  == %f\n", syn_sum);

    //printf("c == %d,  sou = %f, seiyaku = %f\n", ten, sou_mul, syn_mul_send / syn_sum);


    cal[2] += sou_mul * (syn_mul_send / syn_sum);


    return;
}




int main(void){
    int i, j, l, k, m;
    int f_row, f_column, f_pari;
    int kaisu;
    int error;
    int check;
    int pari;
    time_t start_time, end_time;
    double best[3][BEST_G];
    double worst[3][WORST_G];
    int same_kazu[3][BEST_G];
    int sum_ave_loop = 0;



    start_time = time(NULL);

    s_p = 0.9;

    printf("p(1) = %f\n", s_p);


    for (i = 0; i < 3; i++){
        for (j = 0; j < BEST_G; j++){
            best[i][j] = 1.0;
            same_kazu[i][j] = 0;

        }
```

```
}

for (i = 0; i < 3; i++){
    for (j = 0; j < WORST_G; j++){
        worst[i][j] = 0;
    }
}
```

```
FILE *fq;
if ((fq = fopen("kurasu_2.txt", "r")) == NULL){
    printf("\a no open [%d]\n", __LINE__);
    exit(1);
}

fscanf(fq, "%d %d %d", &f_column, &f_row, &f_pari);
p_column = f_column;
p_row = f_row;
pari = f_pari;

printf("p_column = %d\n", p_column);
printf("p_row = %d\n", p_row);


Hon = p_row + p_column - 1;
printf("Hon = %d\n", Hon);

loop = (p_row + p_column) / 2 + 1;
printf("loop = %d\n", loop);


lim = malloc(Hon * sizeof(Par));
if (lim == NULL){
    printf("\a no kakuho [%d]\n", __LINE__);
    exit(1);
}

symbol_dis = malloc(p_column * sizeof(double) * 2);
if (symbol_dis == NULL){
    printf("\a no kakuho [%d]\n", __LINE__);
    exit(1);
}

dis_pari = malloc(pari * sizeof(double) * 3);
if (dis_pari == NULL){
    printf("\a no kakuho [%d]\n", __LINE__);
    exit(1);
}

max = 1 << p_column;



for (i = 0; i < pari; i++){
    printf("%d", i);
    error = 0;
    for (j = 0 ; j < Hon; j++){
        fscanf(fq, "%d %d", &f_column, &f_row);
        lim[j].column = f_column;
        lim[j].row = f_row;
        //printf("%d", lim[j].column);
        //printf(" %d", lim[j].row);
        //putchar('\n');
    }

    //printf("ero = %f\n", (double)error/kaisu);
```

```
for (j = 0; j < max; j++){
    error_count(j);
}

//printf("avaae === %d\n", ave_loop);



for (j = 0; j < 3; j++){
    //printf("erro%d  =  %f\n", j, (double)count[j]);
    dis_pari[i][j] = 1 - cal[j];
    for (k = 0; k < BEST_G; k++){
        if(fabs(1 - cal[j] - best[j][k])<= 0.000000000001){
            same_kazu[j][k] += 1;
            dis_pari[i][j] = best[j][k];
            break;
        }



        if(1 - cal[j]  < best[j][k]){
            for (l = BEST_G - 1; l > k; l--){
                best[j][l] = best[j][l - 1];
                same_kazu[j][l] = same_kazu[j][l - 1];

                for (m = 0; m < Hon; m++){
                    lim[m].best_column[j][l] = lim[m].best_column[j][l - 1];
                    lim[m].best_row[j][l] = lim[m].best_row[j][l - 1];
                }

            }

            best[j][k] = 1 - cal[j];
            same_kazu[j][k] = 1;

            for (l = 0; l < Hon; l++){
                lim[l].best_column[j][k] = lim[l].column;
                lim[l].best_row[j][k] = lim[l].row;
            }

            break;

        }



    }

    for (k = 0; k < WORST_G; k++){
        if(1 - cal[j]  == worst[j][k]){
            break;
        }

        if(1 - cal[j]  > worst[j][k]){
            for (l = WORST_G - 1; l > k; l--){
                worst[j][l] = worst[j][l - 1];

                for (m = 0; m < Hon; m++){
                    lim[m].worst_column[j][l] = lim[m].worst_column[j][l - 1];
                    lim[m].worst_row[j][l] = lim[m].worst_row[j][l - 1];
                }

            }

            worst[j][k] = 1 - cal[j];
```

```
                for (l = 0; l < Hon; l++){
                    lim[l].worst_column[j][k] = lim[l].column;
                    lim[l].worst_row[j][k] = lim[l].row;
                }

                break;

            }



        }
        cal[j] = 0;


    }


    //printf("%d", i);



}

putchar('\n');

FILE *same_num;
if ((same_num = fopen("num_same.txt", "w")) == NULL){
    printf("\a no open [%d]\n", __LINE__);
    exit(1);
}

FILE *same_num_rui;
if ((same_num_rui = fopen("num_same_rui.txt", "w")) == NULL){
    printf("\a no open [%d]\n", __LINE__);
    exit(1);
}

FILE *num_dis_pari;
if ((num_dis_pari = fopen("num_dis_pari.txt", "w")) == NULL){
    printf("\a no open [%d]\n", __LINE__);
    exit(1);
}

int ruiseki[3] = {0, 0, 0};

for (i = 0; i < BEST_G; i++){
    for (j = 0; j < 3; j++){
        if (best[j][i] == 1){
            fprintf(same_num, "\t\t\t");
            fprintf(same_num_rui, "\t\t\t");

        } else {
            ruiseki[j] += same_kazu[j][i];

            fprintf(same_num, "%.20f\t", best[j][i]);
            fprintf(same_num_rui, "%.20f\t", best[j][i]);

            fprintf(same_num, "%d\t", same_kazu[j][i]);
            fprintf(same_num_rui, "%d\t", ruiseki[j]);

            fprintf(same_num, "\t");
            fprintf(same_num_rui, "\t");

        }
    }
    fprintf(same_num, "\n");
    fprintf(same_num_rui, "\n");
```

```
}

for (i = 0; i < 3; i++){
    printf("dec%d == ", i);
    //fprintf(same_num, "%d\t", i + 1);
    for (j = 0; j < BEST_G; j++){
        printf("%d  ", same_kazu[i][j]);
        //fprintf(same_num, "%d\t", same_kazu[i][j]);
    }
    //fprintf(same_num, "\n");
    putchar('\n');
}

for (i = 0; i < pari ;i++){
    for (j = 0; j < 3; j++){
        fprintf(num_dis_pari, "%.20f\t%d\t\t",dis_pari[i][j], i + 1);
    }
    fprintf(num_dis_pari, "\n");
}




FILE *fbest;
if ((fbest = fopen("num_pro_best.txt", "w")) == NULL){
    printf("\a no open [%d]\n", __LINE__);
    exit(1);
}

FILE *fbest_pari;
if ((fbest_pari = fopen("num_pari_best.txt", "w")) == NULL){
    printf("\a no open [%d]\n", __LINE__);
    exit(1);
}

FILE *fworst;
if ((fworst = fopen("num_pro_worst.txt", "w")) == NULL){
    printf("\a no open [%d]\n", __LINE__);
    exit(1);
}

FILE *fworst_pari;
if ((fworst_pari = fopen("num_pari_worst.txt", "w")) == NULL){
    printf("\a no open [%d]\n", __LINE__);
    exit(1);
}

//性能の良い行列と確率の書き出し
for (j = 0; j < 3; j ++){
    fprintf(fbest,"%d\t", j + 1);

    for (k = 0; k < BEST_G; k++){
        fprintf(fbest, "%.20f\t", best[j][k]);
    }

    fprintf(fbest, "\n");
}


//fprintf(fbest_pari, "%d %d %d", p_column, p_row, BEST_G);



for (j = 0; j < 3; j++){
    for (k = 0; k < Hon; k++){
        for (l = 0; l < BEST_G; l++){
            fprintf(fbest_pari, "%d\t%d\t \t", lim[k].best_column[j][l], lim[k].best_row[j][l]);
        }
```

```
            fprintf(fbest_pari, "\n");
        }

        fprintf(fbest_pari, "\n");
    }


    //性能の悪い行列と確率の書き出し
    for (j = 0; j < 3; j ++){
        fprintf(fworst,"%d\t", j + 1);

        for (k = 0; k < BEST_G; k++){
            fprintf(fworst, "%f\t", worst[j][k]);
        }

        fprintf(fworst, "\n");
    }

    //fprintf(fworst_pari, "%d %d %d", p_column, p_row, WORST_G);

    for (j = 0; j < 3; j++){
        for (k = 0; k < Hon; k++){
            for (l = 0; l < WORST_G; l++){
                fprintf(fworst_pari, "%d\t%d\t \t", lim[k].worst_column[j][l], lim[k].worst_row[j][l]);
            }
            fprintf(fworst_pari, "\n");
        }

        fprintf(fworst_pari, "\n");
    }



    fclose(fbest);
    fclose(fbest_pari);

    fclose(fworst);
    fclose(fworst_pari);

    fclose(same_num);
    fclose(same_num_rui);


    fclose(fq);

    fclose(num_dis_pari);


    free(lim);
    free(symbol_dis);
    free(dis_pari);

    end_time = time(NULL);

    /* 計測時間の表示 */
    printf("time:%ld\n",end_time - start_time);
    printf("p(1) = %f\n", s_p);


    //printf ("coco1 time %d", sum_coco1_time);
    return 0;

}
```