信州大学
大学院理工学系研究科

修士論文

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

指導教員　西新 幹彦 准教授

専攻　電気電子工学専攻
学籍番号　15TM203G
氏名　飯島 一貴

2017 年 3 月 5 日

目次

1　はじめに

1.1　研究背景と目的

情報伝達方法の一つとしてパケット通信がある．パケット通信とはコンピュータ通信において、データを小さなまとまりに分割して一つ一つ送受信する通信方式である．分割されたデータはパケットと呼ばれ，多くの場合はメッセージを符号化し，その符号語をパケットに収める．パケット通信を使うと，送受信間の通信に途中の回線が占有されることがなくなり，通信回線を効率よく利用することができる．また，柔軟に経路選択が行えるため，ネットワークの一部に障害が出たとしても他の回線で代替することができる利点もある．

通常のパケット通信ではパケットの中に伝えるべき情報が収められている．しかしながら，Anantharam and Verdú[1] によると，パケット間隔もまた情報を運ぶことができる．つまり，パケットの送信間隔の長短に意味を与えておくことで受信者はパケットの到着間隔から情報を得ることができる．パケット通信を行う際，ネットワーク内では様々な処理時間の変化や転送経路の変動等が発生する．これによりパケットの時間間隔が送信時と受信時で異なってしまい，復号器では誤り訂正が必要になる．

大きさのない空のパケットの送信間隔に情報を載せたときの，単一サーバ待ち行列システムの通信路容量は，文献 [1] で既に求められている．また，Coleman and Kiyavash[2] はパケット間隔を用いた 4 元の離散時間通信路に対して記憶のある通信路モデルを復号器に内蔵することで，シンボル誤り確率が 10^{-4} の桁数で通信路容量のおよそ 82% の符号化レートを達成する実用性のある符号器 · 復号器を作製した．

文献 [2] では，低密度パリティ検査符号（Low Density Parity Check Code, 以下では LDPC 符号と略記する）[3] の復号に利用する sum-product 復号法と通信路のグラフィカルモデルを融合させた復号アルゴリズムを適用することにより，通信路特性を十分に考慮に入れた復号を可能にしている．通信路のグラフィカルモデルは実装に適した簡単化が施されているものの，非常に難解である．したがって，離散時間のパケット間隔を用いた通信路において文献 [2] よりも良い符号器 · 復号器を構成することは容易ではない．

そこで本研究では文献 [2] の手法の複雑さを分解して考察する．まずはじめに，パケット間隔を用いたシステムの通信路を無記憶の通信路とみなす単純化を行う．次に，無記憶とみなした通信路に対してシンボルの送受信シミュレーションを行い，送受信シンボルを計数することで，通信路行列を算出する．そして，得られた通信路行列から有本-Bluhat アルゴリズム [4] を用いて無記憶離散時間通信路としての通信路容量を計算し，文献 [2] で達成された符号化レートと比較する．最後に，得られた通信路行列をもとに，sum - product アルゴリズムを復号器に実装し，シンボル誤り確率を調査する．

以上より，通信路の記憶の有無が復号性能に与える影響を調べ，今後の課題を浮き彫りにすることを本研究の目的とする．

なお，パケット間隔を用いたシステムの最尤復号に関する研究として飯島 [5] がある．

1.2　本論文の構成

本論文は次のような章から成る．2 章では本研究で扱う離散時間通信システムの概要と符号の定義について述べる．3 章では本来記憶をもつパケット間隔の通信システムを半ば強引ではあるが無記憶通信路モデルへと単純化を行い，送受信シンボルとパケット間隔を 1 対 1 に対応させる手法を具体例をまじえて説明する．4 章では一般の無記憶離散通信路における通信路行列の定義と，有本-Bluhat アルゴリズムによる通信路容量の計算方法を説明する．5 章では本研究で構成する LDPC 符号の符号化と復号化の手順を説明する．6 章では無記憶とみなした通信路に対する，通信路容量の観点と，通信路符号化の観点から考察し，今後の展望について述べる．最後の 7 章で本論文をまとめる．

2　通信システム

この章ではパケット間隔を用いたシステムを離散時間の数理モデルを用いて表現し，その概要を述べる．そして時間間隔を用いた符号と，離散時間通信路に対する達成可能性と通信路容量の概念を定義する．

2.1　通信路モデル

インターネットのようなネットワークでは複数の送信者が不規則にパケットを送信する．パケットは複数のサーバに中継されて最終目的地に届けられる．パケットが送信者のもとを出発してから目的地に到着するまでの時間は，他のユーザがどの程度パケットを送信しているか，すなわちネットワークのトラフィックの状態に依存する．いま，一対の送受信者に着目すると，ネットワーク全体はひとつの待ち行列システムとみなすことができる．本来ならパケットの中にも情報を載せるのだが，従来研究 [1][2] に基づき，本研究では問題を簡単化するためにパケットの中の情報量は 0 とし，パケットの送信間隔のみで通信することを考える．このような単一サーバによって構成された FIFO 待ち行列システムのモデルを図 1 に示す．

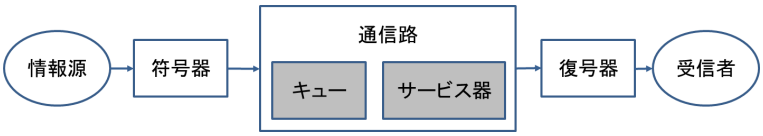

図 1　通信路モデル

符号器では送信対象のメッセージを非負整数のパケット時間間隔の列に変換し，通信路へ送信する．本研究の離散時間システムは整数の時間のみを扱うものとし，議論の基準となる時間の単位を [slot] と表記する．符号器からのパケットの出力過程はそのまま通信路への入力過程となり，入力レート λ[packet/slot] で通信路にパケットが到着するものとする．通信路は単一のキューとサービス器により構成される．サービス器は幾何分布に従う離散時間でサービスを行うと仮定し，サービスレート μ[packet/slot] の性能を持っているとする．単一サーバ待ち行列システムを考えているため，一つのパケットがサービスを受けている途中で到着したパケットを順序を崩すことなく待たせるために，サービス器の手前にキューを設置する．このキューは待機するパケットをすべて格納できるように十分な大きさがあると仮定する．復号器では到着したパケットの時間間隔からメッセージを復号する．この符号器 · 復号器のペアを符号と呼ぶ．

2.2　パケット間隔で情報を送る符号

形式的には，符号語は非負整数の並びである．本研究では文献 [1] に倣い，パケットの間隔を用いた符号を次のように定義する．

定義 1　非負整数ベクトル $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ を符号語という．符号器が符号語 $\boldsymbol{x}$ を送信するとき，時刻 0 で空のキューに対して時刻 $\sum_{i=1}^{k} x_i$ に k 番目 $(k \geq 1)$ の到着が起こる．符号器は M 個の符号語を持っているとし，それらは等確率で選ばれて送信されるとする．復号器は n 個の出発を観測し，正しい符号語を平均確率 $1 - \epsilon$ で復号するとする．最後の出発が起こる時刻の平均を T とおく．このような符号器 · 復号器を (n, M, T, ϵ) 符号という．この符号の符号化レートを $(\log M)/T$ と定める[*1]．したがって，符号化レートとは単位時間あたりに符号器が送信する情報量を表している．

2.3　達成可能性と通信路容量

通信において符号化レートは大きいほうがよいが，復号器で正しく復号されなければ意味がない．したがって，与えられた符号化レートに対してほぼ確実に正しく復号できるような復号

[*1] 本論文を通して log は自然対数を表す．

器が存在するかどうかが重要である．この概念を次のように定義する．以下に示す 2 つの定義は文献 [1] にもとづいて文献 [6] で定義されたものである．

定義 2　符号化レート R が**達成可能**とは

$$\liminf_{n\to\infty} \frac{\log M_n}{T_n} \geq R \tag{1}$$

$$\lim_{n\to\infty} \frac{n}{T_n} > 0 \tag{2}$$

$$\lim_{n\to\infty} \epsilon_n = 0 \tag{3}$$

となるような符号の列 $\{(n, M_n, T_n, \epsilon_n)\}_{n=1}^{\infty}$ が存在することである．通信路容量を

$$C \triangleq \sup\{R| \text{ 符号化レート } R \text{ は達成可能.}\}$$

と定める．

定義 3　符号化レート R が**入力レート λ で ϵ-達成可能**とは

$$\liminf_{n\to\infty} \lambda\frac{\log M_n}{T_n} \geq R \tag{4}$$

$$\lim_{n\to\infty} \frac{n}{T_n} \geq \lambda \tag{5}$$

$$\lim_{n\to\infty} \epsilon_n \leq \epsilon \tag{6}$$

となるような符号の列 $\{(n, M_n, T_n, \epsilon_n)\}_{n=1}^{\infty}$ が存在することである．$0 < \epsilon < 1$ となる任意の ϵ において R が入力レート λ で ϵ-達成可能であれば，R は入力レート λ で達成可能であるという．入力レート λ で達成可能な R の上限を，入力レート λ における通信路容量と呼び，$C(\lambda)$ と表す．すなわち

$$C(\lambda) \triangleq \sup\{R| \text{ 符号化レート } R \text{ は入力レート } \lambda \text{ で達成可能.}\}$$

と定める．

定義 2 と定義 3 には以下の関係がある．

定理 1 サービスレート μ のサーバを持つ通信路に対して C と $C(\lambda)$ は

$$C = \sup_{0<\lambda<\mu} C(\lambda) \tag{7}$$

を満たす．

文献 [2] によると，離散時間のシステムを考えた場合，通信路容量を達成する入力過程はレート λ のベルヌーイ過程であり，入力レート λ における通信路容量 $C(\lambda)$ は

$$C(\lambda) = h(\lambda) - \frac{\lambda}{\mu}h(\mu) \quad [\text{nat/slot}] \tag{8}$$

と表される．ここで，$h(\cdot)$ は e を底とする 2 値エントロピー関数である．式 (8) は λ の関数で，式 (7) に代入して最大値を求めると通信路容量 C は

$$C = \log(1 + e^{-\frac{h(\mu)}{\mu}}) \quad [\text{nat/slot}] \tag{9}$$

となる．このときの入力レート λ は

$$\lambda = \frac{e^{-\frac{h(\mu)}{\mu}}}{e^{-\frac{h(\mu)}{\mu}} + 1} \quad [\text{packet/slot}] \tag{10}$$

となる．

3 無記憶通信路への単純化

キューでパケットが待機すると，それ以後のパケット到着間隔にも影響を及ぼすため，上記のシステムにおける通信路は記憶を持っている．しかし，本研究では単純化のために入力アルファベット数 Q，出力アルファベット数 Q の無記憶通信路とみなして議論を進める．通信路容量を達成する入力過程がベルヌーイ過程のため，本研究では送信するパケット間隔が幾何分布に従うようにパケットを生成して通信路行列を実験的に求める．

この章ではその準備として，幾何分布の生成法と，送受信シンボルとパケット間隔の対応について説明する．

3.1 逆関数法による幾何分布の生成

所望の分布関数 $F_Z(z)$ に従う確率変数 Z は，区間 $[0, 1)$ 上の一様分布に従う確率変数 U を用いて

$$Z = F_Z^{-1}(U) \tag{11}$$

と生成できる．この手法を用いてレート $\lambda \in (0, 1)$ の幾何分布に従う離散確率変数 Z を生成する．幾何分布の分布関数 $F_Z(k)$ は

$$F_Z(k) = 1 - (1 - \lambda)^k \tag{12}$$

で与えられ，一様分布 U から幾何分布 Z は

$$Z = \left\lceil \frac{\log(1 - U)}{\log(1 - \lambda)} \right\rceil \tag{13}$$

と生成することができる．

3.2 送受信シンボルとパケット間隔の対応

送信シンボルを表す確率変数を X，送信パケット間隔を表す確率変数を Z とするとき，これらのとる値を 1 対 1 に対応させる手順を説明する．送信するシンボルは $x \in \{0, 1, \ldots, Q-1\}$ の Q 個の非負整数値の中の 1 つをとる確率変数 X とする．この X を

$$U = \frac{X + \tau}{Q} \tag{14}$$

を用いて分布の変換を行う．τ は区間 $[0, 1)$ 上の任意の実数値とする．もし，X が $\{0, 1, \ldots, Q-1\}$ 上に一様に分布するとすれば，変換後の U も $\{\frac{\tau}{Q}, \frac{1+\tau}{Q}, \ldots, \frac{Q-1+\tau}{Q}\}$ 上に一様に分布する．ここで，十分大きな Q に対して U は区間 $[0, 1)$ 上の一様分布とみなすことができる．

したがって，式 (13) に式 (14) を代入することで，レート λ の幾何分布に従う離散確率変数

$$\begin{aligned} Z &= F_Z^{-1}(U) \\ &= F_Z^{-1}\left(\frac{X + \tau}{Q}\right) \\ &= \left\lceil \frac{\log\left(1 - \frac{(X+\tau)}{Q}\right)}{\log(1-\lambda)} \right\rceil \end{aligned} \tag{15}$$

を得る．この式 (15) を用いて送信するシンボル X に送信パケット間隔 Z を 1 対 1 で対応させ，通信路に送る．送信シンボルと送信パケット間隔の対応の具体例を図 2 に示す．この例では $Q = 4, \lambda = 0.2, \tau = 0.5$ とし，送信シンボル「0」,「1」,「2」,「3」がそれぞれ送信パケット間隔 1[slot]，3[slot]，5[slot]，10[slot] に対応することを表している．

続いて，受信パケット間隔を表す確率変数 $\hat{Z}$ から受信シンボルを表す確率変数 $\hat{X}$ に戻す手順を説明する．無記憶通信路を通して受信された間隔 $\hat{Z}$ は送信間隔 Z と異なっているのが普通である．受信パケット間隔 $\hat{Z}$ を幾何分布の分布関数を用いた式

$$\hat{U} = 1 - (1 - \lambda)^{\hat{Z}} \tag{16}$$

に代入し，再び区間 $[0, 1)$ 上に値をとる確率変数 $\hat{U}$ に変換する．この区間 $[0, 1)$ を $\left[0, \frac{1}{Q}\right)$, $\left[\frac{1}{Q}, \frac{2}{Q}\right)$, $\cdots$, $\left[\frac{Q-1}{Q}, 1\right)$ の均等な Q 個の区間に分割し，それぞれの区間に $0, 1, \cdots, Q-1$ の整数値を対応させる．$\hat{U}$ はこの Q 個の区間のいずれか一つに値をとり，その区間に対応する整数値を $\hat{Z}$ を受けとった時の受信シンボルとする．受信シンボルを表す確率変数 $\hat{X}$ とすると，つまりは

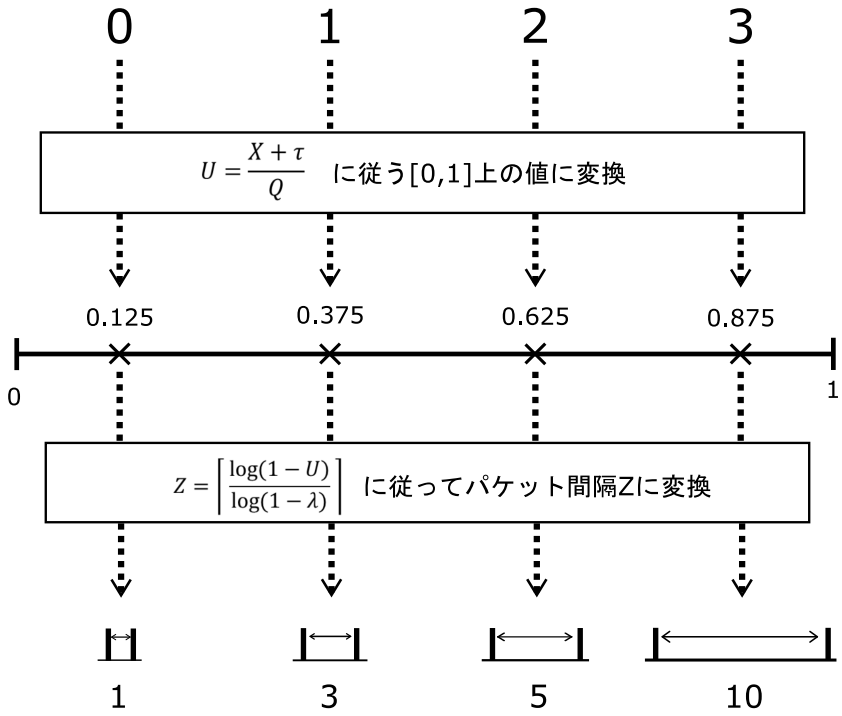

図 2　送信シンボルと送信パケット間隔の対応の具体例

$$\hat{X} = \left\lfloor Q\left(1-(1-\lambda)^{\hat{Z}}\right)\right\rfloor \tag{17}$$

に従って受信シンボル $\hat{X}$ を決定する．

受信パケット間隔と受信シンボルの対応の具体例を図 3 に示す．直前の例と同様に，$Q=4, \lambda=0.2, \tau=0.5$ としている．この例では，受信機は $\hat{Z}=6$[slot] を受信したとする．受信機は式 (16) に従って $\hat{U}=0.7378$ を計算し，区間 $[0.5, 0.75)$ に対応する $\hat{X}=2$ を受信シンボルとして決定する．受信シンボル $\hat{x} \in \{0, 1, \cdots, Q-1\}$ のアルファベットサイズも Q である．以上の手順で，パケット間隔を用いた通信路を無記憶通信路とみなす単純化を行う．

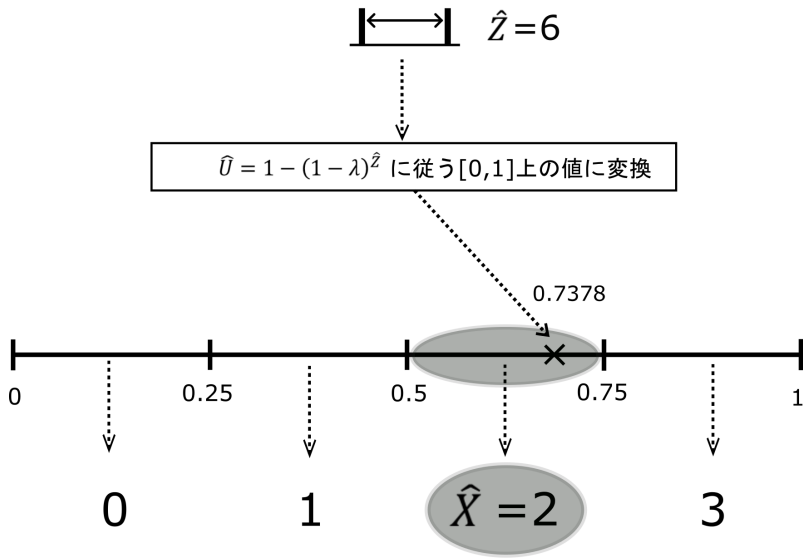

図 3　受信パケット間隔と受信シンボルの対応の具体例

4　通信路行列と通信路容量の算出方法

この章では記憶のない離散通信路の振る舞いを決定する通信路行列の定義と，通信路行列から通信路容量を反復計算によって漸近的に求める有本-Bluhat アルゴリズム [4] について説明する．

4.1　通信路行列

記憶のない入力アルファベットサイズ m 出力アルファベットサイズ n の離散通信路は一般に通信路行列

$$P \triangleq (P_{ji}) = \begin{pmatrix} P_{11} & P_{21} & \dots & P_{m1} \\ P_{12} & P_{22} & \dots & P_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ P_{1n} & P_{2n} & \dots & P_{mn} \end{pmatrix} \tag{18}$$

$$P_{ji} \geq 0, \quad i = 1, 2, \cdots, n, \quad j = 1, 2, \cdots, m, \quad \sum_{j=1}^{m} P_{ji} = 1$$

によって特徴づけられる．本研究における通信路行列は，3.2 節で示した入出力アルファベットサイズ Q，送受信シンボルは $\{0, 1, \ldots, Q-1\}$ の整数値をとる無記憶通信路である．したがって i 行 j 列目の要素には $P_{ji} = \Pr(\hat{X} = j-1 \mid X = i-1)$ を対応させる．

4.2 通信路容量の計算法

通信路に雑音があるとき，信頼できる情報伝送が実現できるための符号化レートの上限として通信路容量が定義される．そして記憶のない通信路の通信路容量 C は相互情報量の入力確率分布に関する最大値として定められる．

n 元確率ベクトル $\boldsymbol{p} = (p_1, p_2, \cdots, p_n)$ の全体の集合 $\mathcal{P}$ を

$$\mathcal{P} \triangleq \left\{ \boldsymbol{p} \;\middle|\; p_i \geq 0 \ (i = 1, 2, \cdots, n),\ \sum_{i=1}^{n} p_i = 1 \right\} \tag{19}$$

と定義する．式 (18) で表される通信路 P に任意の確率分布 $\boldsymbol{p} \in \mathcal{P}$ を持つ情報源 X をつないだ時，入力 X と出力 $\hat{X}$ 間の相互情報量は

$$I(X;\ \hat{X}) = H(X) - H(X|\hat{X}) \tag{20}$$

と表すことができる．$I(X;\ \hat{X})$ は入力確率分布 $\boldsymbol{p} \in \mathcal{P}$ の関数 $I(\boldsymbol{p})$ とみなすことができ，上式の右辺を具体的に記述すると

$$\begin{aligned} I(\boldsymbol{p}) &= I(X;\ \hat{X}) \\ &= H(X) - H(X|\hat{X}) \\ &= \sum_{i=1}^{n} -p_i \log p_i + \sum_{j=1}^{m} \sum_{i=1}^{n} p_i P_{ji} \log \frac{p_i P_{ji}}{\sum_{k=1}^{n} p_k P_{jk}} \end{aligned} \tag{21}$$

となる．通信路容量は相互情報量の入力確率分布に関する最大値なので

$$C = \max_{\boldsymbol{p} \in \mathcal{P}} I(\boldsymbol{p}) \tag{22}$$

と求められる．ここで，次のような集合 Φ を定義する．

$$\Phi \triangleq \left\{ \varphi = (\varphi_{ij}) \;\middle|\; \varphi_{ij} \geq 0,\ \sum_{i=1}^{n} \varphi_{ij} = 1 \right\} \tag{23}$$

そして相互情報量の拡張された概念として関数

$$I(\boldsymbol{p}; \varphi) \triangleq \sum_{i=1}^{n} -p_i \log p_i + \sum_{j=1}^{m} \sum_{i=1}^{n} p_i P_{ji} \log \varphi_{ij} \tag{24}$$

を定義する。通信路容量は $I(\boldsymbol{p};\varphi)$ を用いて

$$C = \max_{\boldsymbol{p}\in\mathcal{P}} I(\boldsymbol{p}) = \max_{\boldsymbol{p}\in\mathcal{P}} \max_{\varphi\in\Phi} I(\boldsymbol{p};\varphi) \tag{25}$$

と表すことができる．この通信路容量の表現式 (25) に基づき，四則演算と対数関数と指数関数の計算の定まった手順を繰り返して通信路容量をだんだんと精度よく求める方法が有本-Bluhat アルゴリズム [4] である．アルゴリズムの詳細は次のようになる．

ステップ 1　初期入力分布 $\boldsymbol{p}^1$ を等確率分布 $\boldsymbol{p}^1 = (p_1^1, \cdots, p_n^1) = (1/n, \cdots, 1/n)$ と設定する．また，反復回数のカウンタとする変数 $N = 1$ とし，最大反復回数を変数 N_{max} に設定する．

ステップ 2　$i = 1, 2, \cdots, n,\ j = 1, 2, \cdots, m$ のすべてのペア (i, j) に対して次の式を利用して φ_{ij}^N を計算する．

$$\varphi_{ij}^N = \frac{p_i^N P_{ji}}{\sum_{k=1}^{n} p_k^N P_{jk}} \tag{26}$$

ステップ 3　$i = 1, 2, \cdots, n$ のすべての i に対して次の式を利用して α_i^N を計算する．

$$\alpha_i^N = \exp\left(\sum_{j=1}^{m} P_{ji} \log \varphi_{ij}^N\right) \tag{27}$$

ステップ 4　$i = 1, 2, \cdots, n$ のすべての i に対して次の式を利用して p_i^{N+1} を計算する．

$$p_i^{N+1} = \frac{\alpha_i^N}{\sum_{k=1}^{n} \alpha_k^N} \tag{28}$$

そして，次の式を利用して $C(N+1, N)$ を計算する．

$$C(N+1, N) = I(\boldsymbol{p}^{N+1}; \varphi^N) \tag{29}$$

$N + 1 < N_{max}$ ならば N をインクリメントしてステップ 2 に戻る．$N + 1 = N_{max}$ ならば $C(N+1, N)$ を出力してアルゴリズムを終了する．

このようにして求められた $C(N+1, N)$ は，最大反復回数 N_{max} を大きくしていくと通信路 P の通信路容量 C に漸近していく．この手法を用いて，本研究では単純化した通信路の通信路容量を求める．

5　誤り訂正符号の構成

文献 [2] で用いられている LDPC 符号は疎なパリティ検査行列により定義される線形符号である．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．本研究ではパリティ検査行列の各行および各列に出現する非零要素の重みがそれぞれ一定値である正則 LDPC 符号を構成する．

5.1　パリティ検査行列に基づく符号化

2 元 m 行 n 列 $(0 < m < n)$ の疎なパリティ検査行列 H を考える．行列 H のすべての行ベクトルに直交する長さ n の Q 元ベクトル $\boldsymbol{c} = (c_1, c_2, \cdots, c_n)$ の集合

$$\mathcal{C} \triangleq \left\{ \boldsymbol{c} \in \mathbb{F}_Q^n \,|\, \boldsymbol{c}H^T = \boldsymbol{0} \right\} \tag{30}$$

を本研究で定義する符号とする．H^T は行列 H の転置を表し，$\mathbb{F}_Q$ は位数 $Q = 2^t$ の有限体を表すものとする．符号化は $\boldsymbol{c}H^T = \boldsymbol{0}$ を満たすことから，連立方程式を解くことにより行う．本研究では，Gallager の構成法 [6] を用いて正則 LDPC 符号の検査行列を構成するが，得られた検査行列は必ずしもフルランクになるとは限らない．そのため，一般にこの符号の符号化率は

$$\tilde{R} \geq \frac{\log Q^{n-m}}{n} \quad [\text{nat/packet}] \tag{31}$$

となる．最後のパケットが復号器に到着する平均時刻を T と置いていたので，

$$T \simeq \frac{n}{\lambda} \quad [\text{slot}] \tag{32}$$

となり[*2]，式 (31) を単位時間当たりの情報量に変形すると

$$R \geq \frac{\lambda \log Q^{n-m}}{n} \quad [\text{nat/slot}] \tag{33}$$

となる．

5.2　LDPC 符号の復号法の詳細

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 列目要素を h_{ij} と表記する．整数の集合 $\{1, 2, \cdots, n\}$ の部分集合 $A(i), B(j)$ を次のよ

[*2] 式 (32) は $\lim_{n\to\infty} T/n = \lambda$ を意味する．

うに定義する．

$$A(i) \triangleq \{j \mid h_{ij} = 1\}, \tag{34}$$

$$B(j) \triangleq \{i \mid h_{ij} = 1\}. \tag{35}$$

すなわち，$A(i)$ は検査行列 H の i 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，$B(j)$ は検査行列 H の j 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 $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)$ を受信したものと仮定する．通信路としては条件付き確率 $\Pr(y \mid x)$ $(y \in \mathcal{Y}, x \in \mathbb{F}_Q)$ で記述される無記憶通信路を仮定する．ここで，$\mathcal{Y}$ は受信アルファベットであり，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．また，検査行列 H の第 i 行に対応する変数の集合 V_i を

$$V_i \triangleq \{x_j \mid j \in A(i)\} \tag{36}$$

で定義する．確率領域 sum-product 復号法 [3] の詳細は次のようになる．

ステップ 1(初期化) $h_{ij} = 1$ を満たすすべてのペア (i, j) に対して $q_{ij}(x_j) = 1/Q$ と初期設定する．また，反復回数のカウンタとする変数 $l = 1$ とし，最大反復回数を変数 l_{max} に設定する．

ステップ 2(行処理) $i = 1, 2, \cdots, m$ の順に $h_{ij} = 1$ を満たすすべてのペア (i, j) に対して次の更新式を利用して $r_{ij}(x_j)$ を更新する．

$$r_{ij}(x_j) = K \sum_{V_i \backslash x_j} I\left[\sum_{l \in A(i)} x_l = 0\right] \prod_{k \in A(i) \backslash j} q_{ik}(x_k) \Pr(y_k | x_k) \tag{37}$$

ここで定数 K は $\sum_{x_j \in \mathbb{F}_Q} r_{ij}(x_j) = 1$ 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 $V_i \backslash x_j$ は x_j を除く V_i に含まれるすべての変数がすべての可能な値（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 $V_i \backslash \{x_j\}$ 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．同様に集合 D から要素 $d \in D$ を取り除いて得られる集合を $D \backslash d$ と表記する．また，表記 $I[condition]$ は $condition$ が真のとき 1，偽のとき 0 を値としてとる関数を意味する．

ステップ 3(列処理) $j = 1, 2, \cdots, n$ の順に $h_{ij} = 1$ を満たすすべてのペア (i, j) に対して，次の更新式を利用して $q_{ij}(x_j)$ を更新する．

$$q_{ij}(x_j) = K' \prod_{k \in B(j) \backslash i} r_{kj}(x_j) \tag{38}$$

ここで，K' は $\sum_{x_j \in \mathbb{F}_Q} q_{ij}(x_j) = 1$ が成り立つように定められる正規化定数とする．

ステップ 4(一時推定語の計算) $j = 1, 2, \cdots, n$ について

$$g_j(x_j) = K'' \Pr(y_j|x_j) \prod_{k \in B(j)} r_{kj}(x_j) \tag{39}$$

を計算し，その後，一時推定シンボル $\hat{x}_j$ を

$$\hat{x}_j = \arg\max_{k \in \mathbb{F}_Q} g_j(k) \tag{40}$$

と定める[*3]．ここで得られる $(\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n)$ を一時推定語と呼ぶ．K'' は $\sum_{x_j \in \mathbb{F}_Q} g_j(x_j) = 1$ が成り立つように定められる正規化定数である．

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 $(\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n)$ が

$$(\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n) H^T = \mathbf{0} \tag{41}$$

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

ステップ 6(反復回数のカウント) パリティ検査を通過せず，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，$(\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_n)$ 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，シンボル誤り確率を検証していく．

6 通信路とその符号化に関する考察

この章では無記憶とみなした通信路の通信路容量を 4 章で述べた手法で求め考察する．次に無記憶とみなした通信路に対して 5 章で述べた通信路符号化を行い，シンボル誤り確率と符号化レートの関係から，通信路の記憶の有無が復号性能に与える影響を考察し，今後の展望について述べる．

[*3] 本研究では，$g_j(x_j)$ を最大にする x_j が複数ある場合は，候補となる x_j の中の最小の値を一時推定シンボルとする．

6.1　無記憶通信路の通信路容量の算出結果

通信路のサービスレートを $\mu = 0.5$，$\tau = 0.5$ と設定する．アルファベットサイズ $Q = 2$ と $Q = 4$ において，送信シンボルを送信パケット間隔に式 (15) を用いて変換し，無記憶通信路に送信する試行を 1000 万回行った．送信シンボルは Q 個の中から等確率に選ぶものとする．そして各受信シンボルを種類ごとに計数し，送信した総シンボル数で割ることで実験的に通信路行列を求めた．そして得られた通信路行列から前述の有本-Bluhat アルゴリズムを用いて，通信路容量を計算した．入力レート λ に対する通信路容量 C の値を図 4 に示す．入力レート λ における通信路容量の最大値は，$Q = 2$ のとき 0.0962[bit/slot] 付近，$Q = 4$ のとき 0.1591[bit/slot] 付近であることが分かった．一方，Coleman and Kiyavash[2] が達成した符号化レートは $Q = 4$ のときに 0.26398[bit/slot] であり，算出した通信路容量はそれの 60% ほどの値となった．さらに，サービスレート $\mu = 0.5$ のサーバを持つ通信路の通信路容量は式 (9) より 0.3219[bit/slot] であり，記憶を考慮しなければ，達成可能な符号化レートの上限が大幅に下がってしまう結果となった．

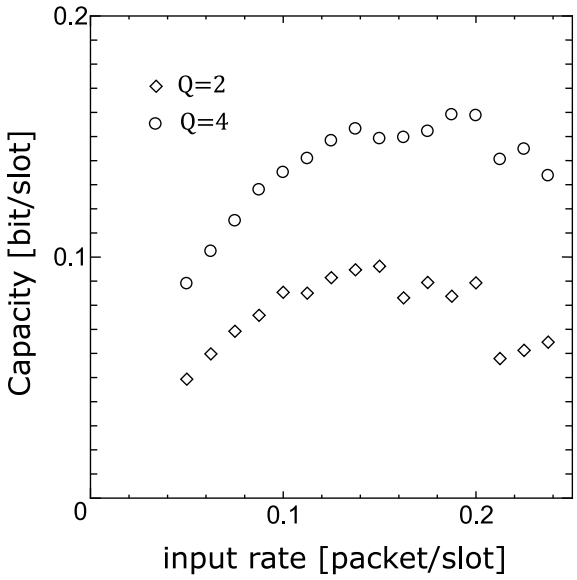

図 4　入力レートに対する通信路容量

6.2　復号性能の評価結果

無記憶とみなした通信路に 4 章で述べた符号を適用する．通信路のサービスレートは $\mu = 0.5$ と固定する．検査行列は符号語長 $n = 900$ を固定し，検査記号数 m を 360，450，540 と変えた 3 種類を用意し，符号語を送信する試行を 100 万回行った．また，検査行列の生成プログラムは和田山 [8] の公開ライブラリを利用した．$Q = 2$ の場合の符号化レートに対するシンボル誤り確率を図 5 に示す．$Q = 2$ のとき，無記憶とみなした通信路容量は 0.0962[bit/slot] であるが，符号化レートをこの 50% 程度まで抑えてようやくシンボル誤り確率 10^{-4} の桁になる．また，式 (9) に基づく通信路容量 C=0.3219[bit/slot] と比較すると，符号化レートをこの 20% 以下に抑えなければシンボル誤り確率 10^{-4} を達成できない結果となった．

次に $Q = 4$ の場合の符号化レートに対するシンボル誤り確率を図 4 に示す．$Q = 4$ のとき，無記憶とみなした通信路容量が 0.1591[bit/slot] であるが，符号化レートをこの 30% 程度まで下げなければシンボル誤り確率が 10^{-4} の桁にならないことが分かった．また，式 (9) に基づく通信路容量と比較すると，符号化レートをこの 18% 以下に抑えなければシンボル誤り確率 10^{-4} を達成できない結果となった．

LDPC 符号は無記憶通信路の符号化においてシャノン限界に迫る性能を発揮すると言われている誤り訂正符号である．しかし，今回のシミュレーション実験の結果では符号化レートが

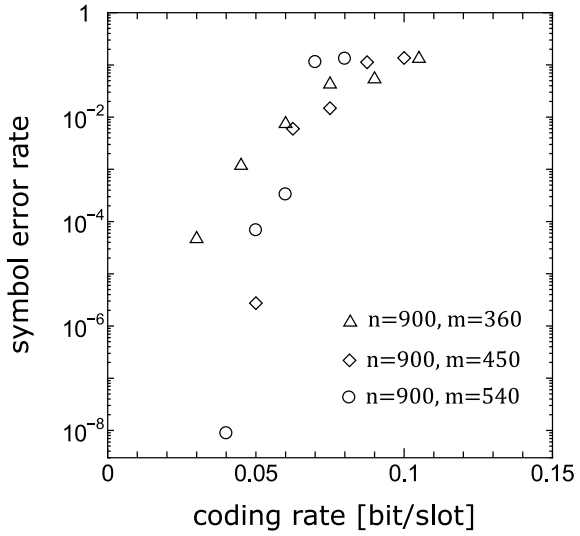

図 5　$Q = 2$ の場合のシンボル誤り確率

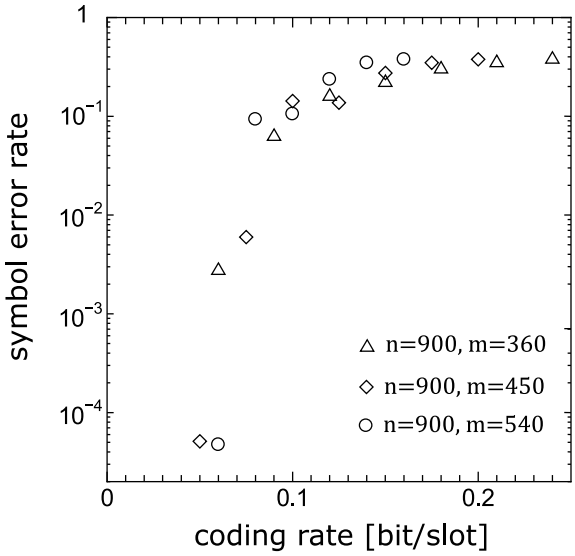

図 6　$Q = 4$ の場合のシンボル誤り確率

大きな範囲では誤り訂正がほとんど行われていなかった．これについて考察をする．符号化レートが大きいと通信路に送信されるパケット間隔が相対的に狭くなる．するとパケットがキューで待つケースが多くなり，隣り合う受信パケット間隔同士の相関に通信路の記憶の側面が強く現れてしまう．しかし本研究では記憶を無視した条件付き確率で表される通信路を仮定し，この条件付き確率から推定シンボルを計算している．したがって，符号化レートが大きい（受信パケット間隔の相関が強い）範囲で通信路の記憶を無視した復号を行ったため，LDPC 符号のもつ誤り訂正能力を超えてしまい，誤り訂正が機能しなくなったと考えられる．一方，Coleman and Kiyavash[2] がシミュレーションを行った LDPC 符号は符号語長 $n = 250$ と非常に短いながらも[*4]シンボル誤り確率が 10^{-4} の桁で通信路容量の 82% の符号化レートをを達成していることから，通信路の記憶に基づいた復号をする方が，性能の良い誤り訂正符号を導入するよりも実用化に向けたアプローチとして妥当であるといえる．

7　まとめ

通信手段の一つとして通信路に送られるパケットの時間間隔に着目し，その間隔によって情報を送ることができると文献 [1] で指摘されている．また，文献 [2] ではアルファベットサイ

[*4] LDPC 符号の誤り訂正能力を十分に発揮するには，符号語長 $n = 250$ では短すぎると思われる．この根拠を付録 A に示す．

ズ $Q = 4$ で，シンボル誤り確率が 10^{-4} の桁数で通信路容量のおよそ 82% の符号化レートを達成する実用性のある符号器 · 復号器を提案している．

本研究では文献 [2] の手法の複雑さを分解し，無記憶とみなした通信路に対して通信路容量を求め，誤り訂正符号の一つである LDPC 符号を構成した．結果として，4 元無記憶離散時間の通信路容量は，文献 [2] が達成した符号化レートの 60% まで減少した．また，構成した誤り訂正符号の復号シミュレーションの結果から，通信路容量から大幅に符号化レートを下げなければ，誤り確率 10^{-4} の桁を達成できないことが分かった．

今後このパケット間隔を用いたシステムの実用化へ向けて，誤り訂正符号の構成に注力するよりも，実装するコスト，規模等を考えながら通信路のモデルを復号器に組み合わせるほうが現実的で堅実なアプローチとなるであろう．

謝辞

本研究を行うにあたり，数多くの助言，指導をしてくださった指導教員西新幹彦准教授，また西新研究室の皆様に感謝の意を表する．

参考文献

[1] Venkat Anantharam, Sergio Verdú, “Bits Through Queues,” IEEE Transactions on Information Theory, vol.42, no.1, pp.4–18, Jan. 1996.

[2] Todd P. Coleman, Negar Kiyavash, “Practical Codes for Queueing Channels: An Algebraic, State-Space, Message-Passing Approach,” IEEE Information Theory Workshop, pp.318–322, 2008.

[3] 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.

[4] 有本卓，情報理論，共立出版， 1976.

[5] 飯島一貴，「パケット間隔で情報を送る通信路に対する最尤復号のための距離関数」，信州大学工学部，学士論文，2015.

[6] 西新幹彦，藤山直貴，「パケット間隔で情報を送る通信路の悲観的な符号化について」，第 35 回情報理論とその応用シンポジウム (SITA2012)，pp.590–595, Dec. 2012.

[7] R.G.Gallager, Low density parity check codes, Cambridge, MIT Press, 1963.

[8] 和田山正，LDPC 符号の公開プログラム，
https://dl.dropboxusercontent.com/u/1820240/supportpages/LDPCpublic.html
（閲覧日 2016/12/7）

付録 A　符号語長の短い LDPC 符号の復号特性とその考察

6.2 節で LDPC 符号の復号性能を十分に発揮するには符号語長 $n = 250$ では短いと主張したが，その根拠を以下に示す．図 7 に符号語長 $n = 252$，検査記号数 $m = 126$ の LDPC 符号と符号語長 $n = 900$，検査記号数 $m = 450$ の LDPC 符号の復号性能を比較したシミュレーション結果を示す．それぞれ文献 [2] の符号語長と本研究で用いた符号語長に対応させている．通信路は二元対称通信路を仮定し，グラフの横軸は通信路のビット反転確率を表している．

通信路のビット反転確率を小さくしていくと，$n = 900$ の LDPC 符号の BER は通信路のビット反転確率から大きく改善されていく傾向にある．一方 $n = 252$ の LDPC 符号の BER は通信路のビット反転確率からあまり改善されていかない．通信路のビット反転確率が 0.02 のとき，$n = 900$ の LDPC 符号の BER は 10^{-9} の桁，$n = 252$ の LDPC 符号の BER は 10^{-3} の桁であり，復号性能に大きな差が生じた．この結果から，文献 [2] で用いられている $n = 250$ の LDPC 符号は，無記憶通信路において十分な誤り訂正能力を発揮するには短い符号語長であると考えられる．

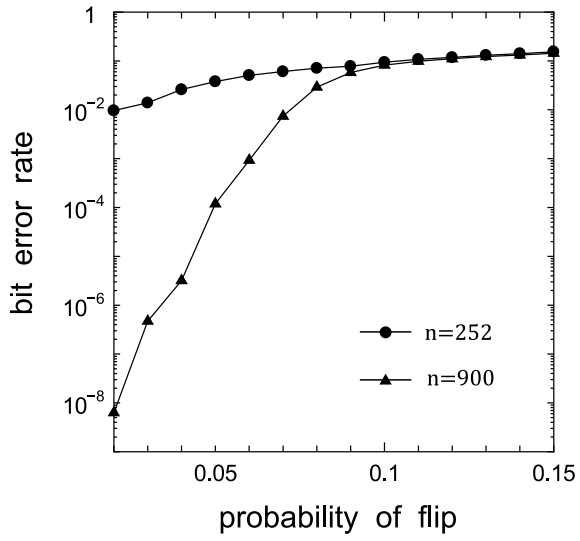

図 7　二元対称通信路に対する LDPC 符号の復号特性

付録 B　ソースコード

送受信シンボル数を 2 次元配列で読み込み，通信路行列と通信路容量を計算するプログラムのソースコードを B.1 に示す．次に，通信路行列と検査行列を 2 次元配列で読み込み，符号語を生成し，送受信シミュレーションによって得られた受信語から sum-product 復号法によってシンボル誤り確率を計算するプログラムのソースコードを B.2 に示す．

B.1　有本-Bluhat アルゴリズムによって通信路容量を計算するプログラム

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define LOOP 2000   /*反復回数*/
#define ROW 4       /*入力シンボル数*/
#define COLUMN 4    /*出力シンボル数*/


int main(int argc, char *argv[]){

    FILE *fp;

    char filename[128];
    int sample[ROW][COLUMN];
    int total[ROW];
    double pmatrix[ROW][COLUMN];    /*通信路行列*/
    double phi[COLUMN][ROW];
    double p[ROW];                  /*入力分布*/
    double alpha[ROW];
    double sum1 = 0;
    double sum2 = 0;
    double sum3 = 0;
    double sum4 = 0;
    double sum5 = 0;

    if(argc != 2){
        printf("使用法： arimoto ファイル名\n");
        exit(1);
    }

    /*初期化*/
    int i,j,k,n;

    for(i=0;i<ROW;i++){
        total[i] = 0;
        alpha[i] = 0;
        p[i] = (1.0/ROW);

        for(j=0;j<COLUMN;j++){
            sample[i][j] = 0;
            pmatrix[i][j] = 0;
            phi[j][i] = 0;
        }
    }

    sprintf(filename, "%s.txt", argv[1]);

    if((fp = fopen(filename, "r")) == NULL) {
```

```
    printf("ファイルをオープンできません.\n");
    exit(1);
}

/*ファイルから読み込み*/
for(i=0; i<ROW; i++){
    for(j=0; j<COLUMN; j++){
        fscanf(fp,"%d", &(sample[i][j]));
    }
}
fclose(fp);

/*通信路行列を計算*/
for(i=0; i<ROW; i++){
    for(j=0; j<COLUMN; j++){
        total[i] += sample[i][j];
    }
}
for(i=0; i<ROW; i++){
    for(j=0; j<COLUMN; j++){
        pmatrix[i][j] = (double)sample[i][j]/total[i];
    }
}


/*反復計算開始*/
for(n=0; n<LOOP; n++){
    for(i=0; i<ROW; i++){
        for(j=0; j<COLUMN; j++){
            sum1 = 0;
            for(k=0; k<ROW; k++){
                sum1 += p[k] * pmatrix[k][j];
            }
            phi[j][i] = (p[i] * pmatrix[i][j]) / sum1;
        }
    }
    for(i=0; i<ROW; i++){
        sum2 = 0;
        for(j=0; j<COLUMN; j++){
            if(pmatrix[i][j] == 0.0){
                sum2 += 0.0;
            }
            else{
                sum2 += pmatrix[i][j] * log(phi[j][i]);
            }
        }
        alpha[i] = exp(sum2);
    }
    for(i=0; i<ROW; i++){
        sum3 = 0;
        for(k=0; k<ROW; k++){
            sum3 += alpha[k];
        }
        p[i] = alpha[i] / sum3;
    }

    /*相互情報量の計算*/

    sum4 = sum5 = 0;

    for(i=0; i<ROW; i++){
        sum4 += -p[i] * log(p[i]);
    }
    for(j=0; j<COLUMN; j++){
        for(i=0; i<ROW; i++){
            if(pmatrix[i][j] == 0.0){
                sum5 += 0.0;
            }
            else{
```

```
                    sum5 += p[i] * pmatrix[i][j] * log(phi[j][i]);
                }
            }
        }
    }

    /*相互情報量を表示 (nat,bit)*/
    printf("C = %f[nat]\n",sum4 + sum5);
    printf("C = %f[bit]\n",(sum4 + sum5)/log(2.0));
    printf("\n");

    /*入力分布*/
    printf("入力分布\n");
    for(i=0; i<ROW; i++){
        printf("p%d===>%f\n",i+1,p[i]);
    }

    return (0);
}
```

B.2 確率領域 sum-product 復号法によって シンボル誤り確率を計算するプログラム

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>

#define MRND 1000000000L
static int jrand;
static long ia[56];

#define RATE_LAMBDA 0.2         /*入力レート設定*/
#define RATE_MU 0.5             /*サービスレート設定*/
#define Q 4                     /*アルファベットサイズ設定*/
#define TAU 0.5                 /*tau 設定*/
#define N 900                   /*検査行列の列数設定*/
#define M 540                   /*検査行列の行数設定*/
#define LOOP_TRANSMIT 1000000   /*符号語送信回数設定*/
#define LOOP_SUM_PRODUCT 30     /*sum-product 復号の反復回数設定*/

int v[N];
int h[M][N];
int parity[M][N];
int code_word[N];
int rcv_word[N];
int estimated_word[N];
int sample[Q][Q];
int total[Q];
double prob[Q][Q];
double rfx[Q][M][N];
double qxf[Q][M][N];
double g[Q];
double cc[Q];
double ss[Q];
double gg[Q];

int spindex;
int row_waight;
int column_waight;
int row;
int column;
```

```
/*一様乱数の生成*/
static void irn55(void)
{
    int i;
    long j;

    for (i = 1; i <= 24; i++){
        j = ia[i] - ia[i + 31];
        if (j < 0) j += MRND;
        ia[i] = j;
    }
    for (i = 25; i <= 55; i++){
        j = ia[i] - ia[i - 24];
        if (j < 0) j += MRND;
        ia[i] = j;
    }
}

void init_rnd(unsigned long seed)
{
    int i, ii;
    long k;

    ia[55] = seed;
    k = 1;
    for (i = 1; i <= 54; i++){
        ii = (21 * i) % 55;
        ia[ii] = k;
        k = seed - k;
        if (k<0) k += MRND;
        seed = ia[ii];
    }
    irn55(); irn55(); irn55();
    jrand = 55;
}

long irnd(void)
{
    if (++jrand>55){ irn55(); jrand = 1; }
    return ia[jrand];
}

double rnd(void)
{
    return (1.0 / MRND)*irnd();
}



/*[0,Q-1) の一様分布*/
int qrand(void)
{
    return((int)(rnd() * Q));
}


/*U を生成*/
double urand(int b)
{
    return((b + TAU) / Q);
}


/*サーバーの役割*/
double rand_mu(void)
```

```
{
    return (ceil((log(1 - rnd()) / log(RATE_MU))));
}


/*送信器の役割*/
double rand_lambda(double c)
{
    return (ceil((log(1 - c) / log(1 - RATE_LAMBDA))));
}


/*パケット間隔から [0,1) 上の値に変換*/
double uni(double interval)
{
    return (1 - pow((1 - RATE_LAMBDA), interval));
}


/*通信路行列を計算*/
void gen_prob_mat(int sample[Q][Q], double prob[Q][Q])
{
    int i, j;
    double total[Q];

    for (i = 0; i<Q; i++){
        total[i] = 0.0;
    }
    for (i = 0; i<Q; i++){
        for (j = 0; j<Q; j++){
            total[i] += sample[i][j];
        }
    }
    for (i = 0; i<Q; i++){
        for (j = 0; j<Q; j++){
            prob[i][j] = (double)sample[i][j] / total[i];
        }
    }
    return;
}


/*検査行列のコピー*/
void matcopy(int h[M][N], int parity[M][N])
{
    int i, j;
    for (i = 0; i<M; i++){
        for (j = 0; j<N; j++){
            parity[i][j] = h[i][j];
        }
    }
    return;
}


/*生成行列を計算*/
int infbit(int v[N], int h[M][N])
{
    int i, j, k;
    int tmp;

    for (i = 0; i < M; i++){
        for (j = i; j < M; j++){
            if (h[j][i] != 0) break;
        }
        if (j >= M){
            for (j = i + 1; j < N; j++){
                if (h[i][j] != 0) break;
            }
```

```
            if (j >= N){
                printf("%d 行目が一次独立ではない\n", i + 1);
                continue;
                exit(1);
            }
            for (k = 0; k < M; k++){
                tmp = h[k][i];
                h[k][i] = h[k][j];
                h[k][j] = tmp;
            }
            tmp = v[i];
            v[i] = v[j];
            v[j] = tmp;
            j = i;
        }
        if (i != j){
            for (k = 0; k < N; k++){
                h[i][k] ^= h[j][k];
            }
        }
        for (j = 0; j < M; j++){
            if (j != i && h[j][i] != 0){
                for (k = 0; k < N; k++){
                    h[j][k] ^= h[i][k];
                }
            }
        }
    }
    return(0);
}


/*符号語生成*/
int gen_codeword(int cw[N], int v[N], int h[M][N])
{
    int i, j;

    for (i = 0; i < M; i++){
        if (h[i][i] == 0){
            cw[v[i]] = (qrand());
        }
    }
    for (; i < N; i++){
        cw[v[i]] = (qrand());
    }
    for (i = 0; i < M; i++){
        if (h[i][i] == 0) continue;
        cw[v[i]] = 0;
        for (j = i + 1; j < N; j++){
            cw[v[i]] ^= h[i][j] * cw[v[j]];
        }
    }
    return(0);
}


/*パリティチェック関数 通過 0 不通 1*/
int parity_check(int estimation[N], int parity_check_mat[M][N])
{
    int i, j;
    int sum;
    for (i = 0; i<M; i++){
        sum = 0;
        for (j = 0; j<N; j++){
            sum ^= estimation[j] * parity_check_mat[i][j];
        }
        if (sum != 0)    return(1);
    }
    return(0);
```

```
}


/*シンボルエラーをカウント*/
int symbol_error_check(int send[], int recv[])
{
    int j;
    int error = 0;
    for (j = 0; j<N; j++){
        if (send[j] != recv[j]){
            error++;
        }
    }
    return (error);
}


/*送受信処理*/
void transmit(int n, int send_word[], int rcv_word[])
{
    int i = 0;
    int j = 0;
    int k;
    double current_time = 0.0;
    double service_end = 0.0;
    int customer_num = 0;
    double sendtime[N];
    double rcvtime[N];
    double rcvepoch[N];

    for (k = 0; k<N; k++){
        sendtime[k] = 0.0;
        rcvtime[k] = 0.0;
        rcvepoch[k] = 0.0;
    }
    sendtime[0] = rand_lambda(urand(send_word[0]));
    for (k = 1; k<N; k++){
        sendtime[k] = sendtime[k - 1] + rand_lambda(urand(send_word[k]));
    }
    for(k=0; k<N; k++){
    }
    while (i < n || customer_num > 0){
        if (customer_num == 0 || (i < n && sendtime[i] < service_end)){
            current_time = sendtime[i];
            i++;
            if (customer_num >= 1){
                customer_num++;
            }
            else {
                service_end = current_time + rand_mu();
                customer_num = 1;
            }
        }
        else {
            current_time = service_end;
            rcvtime[j] = service_end;
            j++;
            customer_num--;
            if (customer_num >= 1){
                service_end = current_time + rand_mu();
            }
        }
    }
    rcvepoch[0] = rcvtime[0];
    for (k = 1; k<N; k++){
        rcvepoch[k] = rcvtime[k] - rcvtime[k - 1];
    }
    for (k = 0; k<N; k++){
        rcv_word[k] = (int)(Q * uni(rcvepoch[k]));
```

```
    }
    return;
}


/*****************************************************************************/

int main(int argc, char *argv[]){

    init_rnd(123456);

    FILE *probfp, *spmatfp;
    int i, j, n;

    char spmatfile[128];
    char probfile[128];

    int loop_transmit;
    int loop_sum_product;
    int count1 = 0;
    int count2 = 0;

    int k, m, l, q;
    int length;
    int sum;
    int max1;
    double max2;
    int likely_word;
    double check_sum[Q];
    int index[N];
    double normal_const;

    for (j = 0; j<N; j++){
        v[j] = j;
    }

    for (i = 0; i<M; i++){
        for (j = 0; j<N; j++){
            h[i][j] = 0;
            parity[i][j] = 0;
        }
    }

    if (argc != 3){
        printf("使用法： LDPC spmat prob\n");
        exit(1);
    }

    sprintf(spmatfile, "%s.txt", argv[1]);
    sprintf(probfile, "%s.txt", argv[2]);

    if ((spmatfp = fopen(spmatfile, "r")) == NULL) {
        printf("ファイルをオープンできません.\n");
        exit(1);
    }

    fscanf(spmatfp, "%d", &row);
    fscanf(spmatfp, "%d", &column);
    fscanf(spmatfp, "%d", &row_waight);
    fscanf(spmatfp, "%d", &column_waight);

    for (i = 0; i<M; i++){
        for (j = 0; j<row_waight; j++){
            fscanf(spmatfp, "%d", &spindex);
            h[i][spindex - 1] = 1;
        }
    }
    fclose(spmatfp);
```

```
if ((probfp = fopen(probfile, "r")) == NULL) {
    printf("ファイルをオープンできません.\n");
    exit(1);
}
for (i = 0; i<Q; i++){
    for (j = 0; j<Q; j++){
        fscanf(probfp, "%d", &(sample[i][j]));
    }
}
fclose(probfp);

gen_prob_mat(sample, prob);
matcopy(h, parity);
infbit(v, h);

loop_transmit = 0;

while (loop_transmit < LOOP_TRANSMIT){

    gen_codeword(code_word, v, h);

    if (parity_check(code_word, parity) == 1){
        printf("生成語が符号語になっていません. \n");
    }


    /*パケット送受信*/
    transmit(row, code_word, rcv_word);


    /*sum_product 復号*/

    for (i = 0; i<M; i++){
        for (j = 0; j<N; j++){
            for (l = 0; l<Q; l++){
                rfx[l][i][j] = 0.0;
                if (parity[i][j] == 0){
                    qxf[l][i][j] = 0;
                }
                else{
                    qxf[l][i][j] = 1.0 / Q;
                }
            }
        }
    }

    loop_sum_product = 0;

    while (loop_sum_product < LOOP_SUM_PRODUCT){

        /*チェックノード処理：qxf を用いて rfx を更新*/
        for (i = 0; i<M; i++){
            for (j = 0; j<N; j++){
                if (parity[i][j] == 0)    continue;
                for (l = 0; l<Q; l++){
                    check_sum[l] = 0.0;
                }
                length = 0;
                for (q = 0; q<N; q++){
                    if ((q == j) || (parity[i][q] == 0))    continue;
                    index[length] = q;
                    length++;
                }
                max1 = 1 << (length * (int)log2((double)Q));
                for (m = 0; m<max1; m++){
                    for (l = 0; l<Q; l++){
                        ss[l] = 1.0;
                    }
                    for (l = 0; l<Q; l++){
```

```
                sum = 0;
                for (n = 0; n<length; n++){
                    sum ^= (m >> (n * (int)log2((double)Q))) & (Q-1);
                }
                if ((sum ^ l) != 0)    continue;
                for (n = 0; n<length; n++){
                    ss[l]
                    *=
                    qxf[(m >> (n * (int)log2((double)Q))) & (Q-1)][i][index[n]] *
                    prob[rcv_word[index[n]]][(m >> (n * (int)log2((double)Q))) & (Q-1)];
                }
                check_sum[l] += ss[l];
            }
        }
        normal_const = 0.0;
        for (l = 0; l<Q; l++){
            normal_const += check_sum[l];
        }
        for (l = 0; l<Q; l++){
            rfx[l][i][j] = check_sum[l] / normal_const;
        }
    }
}


/*変数ノード処理：rfx を用いて qxf を更新*/
for (j = 0; j<N; j++){
    for (i = 0; i<M; i++){
        if (parity[i][j] == 0)    continue;
        for (l = 0; l<Q; l++){
            cc[l] = 1.0;
        }
        for (k = 0; k<M; k++){
            if ((k == i) || (parity[k][j] == 0))    continue;
            for (l = 0; l<Q; l++){
                cc[l] *= rfx[l][k][j];
            }
        }
        normal_const = 0.0;
        for (l = 0; l<Q; l++){
            normal_const += cc[l];
        }
        for (l = 0; l<Q; l++){
            qxf[l][i][j] = cc[l] / normal_const;
        }
    }
}

/*一時推定語の計算*/
for (j = 0; j<N; j++){
    for (l = 0; l<Q; l++){
        gg[l] = 1.0;
        for (i = 0; i<M; i++){
            if (parity[i][j] == 0)    continue;
            gg[l] *= rfx[l][i][j];
        }
        gg[l] *= prob[rcv_word[j]][l];
    }
    normal_const = 0.0;
    for (l = 0; l<Q; l++){
        normal_const += gg[l];
    }
    for (l = 0; l<Q; l++){
        g[l] = gg[l] / normal_const;
    max2 = 0.0;
    for (l = 0; l<Q; l++){
        if (max2 < g[l]){
            max2 = g[l];
            likely_word = l;
```

```
                    }
                }
                estimated_word[j] = likely_word;
            }

            /*パリティ検査*/
            if ((parity_check(estimated_word, parity)) == 0)    break;
            loop_sum_product++;
        }

        count1 += symbol_error_check(code_word, estimated_word);
        count2 += symbol_error_check(code_word, rcv_word);

        loop_transmit++;
    }

    printf("==============\n");
    printf("検査行列 (%d, %d), 行重み %d, 列重み %d\n", row, column, row_waight, column_waight);
    printf("訂正前 シンボル誤り %d / %d\n", count2, N * LOOP_TRANSMIT);
    printf("訂正後 シンボル誤り %d / %d\n", count1, N * LOOP_TRANSMIT);

    return(0);
}
```