信州大学工学部

学士論文

パケット間隔で情報を送る通信路の記憶を考慮した Sum-Product 復号法の検証

指導教員　西新 幹彦 准教授

学科　電子情報システム工学科
学籍番号　17T2076J
氏名　佐藤 奏杜

2021 年 3 月 3 日

目次

1　はじめに

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その一つ一つを送受信するという通信方法である．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，通信回路を効率よく使用することが出来る．また，通信経路の選択をすることが出来るのでネットワーク上一部に障害が生じてしまったとしても他の経路を利用し通信を行うことが出来るという利点もある．

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] は，パケット間隔が情報を運ぶことができると指摘している．つまりパケットの送信間隔の長さに意味を与えることで受信者は到着するパケットの間隔で情報を得ることができるのである．パケット通信を行う際は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移動などが発生してしまい，パケットの時間間隔が送信時と受信時とで異なってしまうため復号器での誤り訂正が必要とされる．そして，誤り訂正された間隔から元となる情報を復号することで正しく情報を伝達することが可能とされているのである．

大きさのない空のパケットの送信間隔に情報を与えた時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．文献 [2] で Coleman and Kiyavash 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，通信路容量に近い符号化レートを達成する実用性のある符号器，復号器を設計しており，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）と Sum-Product 復号法を用いることで，通信路特性を十分に考慮した復号を可能にしている．しかし，文献 [2] では，手法の詳細は記述されていない．それを受け，文献 [3] では，文献 [2] で述べられていることをもとに，手法を再現するのに足りない詳細部分を独自に補い，具体的な手法を明確にした．

文献 [3] では性能評価に至っていないため，本研究ではその手法をもとに作成した符号に対して送受信シミュレーションを行うことで基本的な動作を確認する．また，シンボルの復号誤り確率の測定も行う．さらに，副次的な研究として，復号するまでにかかった Sum-Product 復号処理におけるループの回数を比較することにより復号性能を検証する．

2　通信路モデルと符号化定理

本章では，パケット間隔を用いた通信システムを連続時間の数理モデルを用いて表現し，概要を述べる．本研究では文献 [4] で提示された通信路モデルを利用する．

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それぞれの受信者にパケットを不規則に送信する．各パケットは複数のサーバを通り最終的な目的地に到着する．パケットが送信者から出発してから目的地に到着するまでの時間は，ほかのユーザが送信しているパケットの数，すなわちネットワーク上での混み具合などに依存する．ここで，一組の送受信者に着目して考えるとネットワーク全体は複雑な待ち行列システムとして表される．本研究では簡単化のため複雑な待ち行列システムをシンプルな単一サーバ FIFO 待ち行列システムとみなす．そのため，サービスを受けている最中に到着したパケットの順序を崩さないためにサーバの前に十分大きなキューがあると仮定する．また，サーバはパラメータ μ の指数分布に従う処理時間でパケットを処理する．

このような待ち行列システムによる通信路モデルを図 1 に示す．

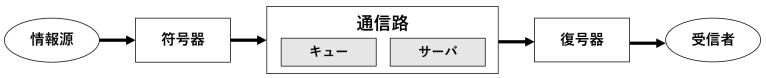

図 1　通信路モデル

通常，パケット通信はパケットの中に情報を入れて通信を行うのに対して，本研究では，パケットの送信間隔を用いて情報を伝達することを目的としている．そのため，パケットには情報を入れず，パケットの送信間隔のみで通信を行う．すなわち，符号器では，送信されるメッセージを時間間隔の列に変換し通信路へと送信する．そして，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．

2.2 通信路符号化定理

形式的には，符号語は非負実数の並びである．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　符号語は非負実数ベクトル $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ として表される．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．復号器にとっての受信語とは，サーバからの出発間隔である．復号器の復号誤り確率を ϵ とおく．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．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．なお本論文では log は自然対数を表す．

通信において，符号化レートが大きいことが求められると同時に復号器で正しく復号されるこ

とも重要とされている．したがって，与えられた符号化レートに対して正しく復号できるような復号器を設計しなければならない．この性質ついて文献 [1] に基づいて文献 [4] で次のように定義されている．

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 $\{(n, M_n, T_n, \epsilon_n)\}_{i=1}^{\infty}$ 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 $\{(n, M_n, T_n, \epsilon_n)\}_{i=1}^{\infty}$ 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** という．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．

上記の定義のもとで次のような 2 つの定理が成り立つことが知られている．

定理 1

$$C(\lambda) = \lambda \log \frac{\mu}{\lambda},\ \lambda \leq \mu \tag{7}$$

定理 2

$$C = \max_{\lambda\leq\mu} C(\lambda) = e^{-1}\mu \tag{8}$$

3　基本的な Sum-Prodcut 復号法

本研究は，通信路に記憶がない場合における Sum-Product 復号法をもとに，記憶を考慮した場合の手法を明確にしていく．本章では，無記憶通信路における具体的な例を用いて基本的な Sum-Prodcut 復号法の手順を示す．

3.1　無記憶通信路と尤度関数

定常無記憶通信路における基本的な Sum-Product 復号の計算手順について，具体的な符号を用いて説明していく．

符号シンボル x が送信された際に受信シンボル y が現れる確率は，

$$W(y|x) = P\{Y = y|X = x\} \tag{9}$$

と表すことが出来る．

この通信路に対して，長さ 5 の符号語を送信する場合は通信路を独立に 5 回使うと考えればよい．ここで，$\boldsymbol{x}$= $(x_1, x_2, \ldots, x_5)$, $\boldsymbol{y}$= $(y_1, y_2, \ldots, y_5)$ とすると，長さ 5 の符号語において，符号語 $\boldsymbol{x}$ が決定したもとで受信語 $\boldsymbol{y}$ が現れる確率は

$$W^5(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1,2,3,4,5} W(y_i|x_i) \tag{10}$$

と表される．

通信路に入力される符号語は一定の制約を満たしたものの中から選ばれる．本研究ではひとつの小さい符号を対象にして考察を進める．具体的には，符号語長が 5 で

$$x_1 + x_2 = 0 \tag{11}$$
$$x_2 + x_3 + x_4 = 0 \tag{12}$$
$$x_4 + x_5 = 0 \tag{13}$$

の制約を満たすものを符号語とする．この制約は行列を用いて

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = 0 \tag{14}$$

と表すことができる．このとき式 (14) 左辺の 3×5 行列をパリティ検査行列という．また，シンボルは 0, 1, 2, 3 の 4 種類とする．

一般に通信路の入力と出力は異なっているので，通信の出力側に設置された復号器は受信語を観測したもとで，実際に送信された符号語を推定しなければならない．受信語 $\mathbf{y}$ を受信したもとで条件付き確率 $P\{X^5 = \mathbf{x}|Y^5 = \mathbf{y}\}$ が最大となるような符号語 x を選択する復号方法を最尤復号という．この条件付確率は，

$$P\{X^5 = \boldsymbol{x}|Y^5 = \boldsymbol{y}\} = \frac{P\{X^5 = \boldsymbol{x}, Y^5 = \boldsymbol{y}\}}{P\{Y^5 = \boldsymbol{y}\}} \tag{15}$$

$$= \frac{P\{X^5 = \boldsymbol{x}\} \cdot P\{Y^5 = \boldsymbol{y}|X^5 = \boldsymbol{x}\}}{P\{Y^5 = \boldsymbol{y}\}} \tag{16}$$

と表すことができる．すると，$\boldsymbol{y}$ は観測された値であるから $P\{Y^5 = \boldsymbol{y}\}$ は定数となる．また，符号語は等確率で送信されるとすれば，$P\{X^5 = \boldsymbol{x}\}$ も定数となる．したがって，$P\{X^5 = \boldsymbol{x}|Y^5 = \boldsymbol{y}\}$ を最大化する $\boldsymbol{x} \in \boldsymbol{C}$ を求めるためには，$W^5(y|x)$ を最大化するような $\boldsymbol{x} \in \boldsymbol{C}$ を求めればよい．

最尤復号を実際に行おうとするとすべての符号語に対して条件付確率を計算しなければならない．この計算量を削減するために，シンボル単位で最尤推定する方法が知られている．この方法はシンボル MAP 復号と呼ばれている．例えば受信語 $\boldsymbol{y}$ に対して送信シンボル x_1 を推定する際には条件付確率 $P\{X_1 = x_1|Y^5 = \boldsymbol{y}\}$ を最大化する x_1 を求めればよい．ここで条件付確率は先ほどと同様に，

$$P\{X_1 = x_1|Y^5 = \boldsymbol{y}\} = \sum_{x_2,x_3,x_4,x_5} P\{X^5 = x_1 \dots x_5|Y^5 = \boldsymbol{y}\} \tag{17}$$

$$= \sum_{x_2,x_3,x_4,x_5} \mathbf{1}\{\boldsymbol{x} \text{ が符号語 }\} P\{X^5 = x_1 \dots x_5|Y^5 = \boldsymbol{y}\} \tag{18}$$

$$= \sum_{x_2,x_3,x_4,x_5} \mathbf{1}\{\boldsymbol{x} \in \boldsymbol{C}\} \frac{P\{X^5 = \boldsymbol{x}\} P\{Y^5 = \boldsymbol{y}|X^5 = \boldsymbol{x}\}}{P\{Y^5 = \boldsymbol{y}\}} \tag{19}$$

$$= \frac{1}{|\boldsymbol{C}|} \frac{1}{P\{Y^5 = \boldsymbol{y}\}} \sum_{x_2,x_3,x_4,x_5} f_1(x_1, x_2) f_2(x_2, x_3, x_4) f_3(x_4, x_5) \prod_{i=1}^{5} W(y_i|x_i) \tag{20}$$

となる．ただし，符号語であるための制約を $f_1(x_1, x_2) f_2(x_2, x_3, x_4) f_3(x_4, x_5)$ とおいた．$|\boldsymbol{C}|$ と $P\{Y^5 = \boldsymbol{y}\}$ は定数なので最大化問題を解く際には関係がない．したがって，

$$\sum_{x_2,x_3,x_4,x_5} f_1(x_1, x_2) f_2(x_2, x_3, x_4) f_3(x_4, x_5) \prod_{i=1}^{5} W(y_i|x_i) \tag{21}$$

を最大化するような x_1 を求めればよい．この式はさらに，

$$
\begin{aligned}
&\sum_{x_2,x_3,x_4,x_5} f_1(x_1,x_2)f_2(x_2,x_3,x_4)f_3(x_4,x_5)\prod_{i=1}^{5}W(y_i|x_i)\\
&= W(y_1|x_1)\\
&\times \sum_{x_2} f_1(x_1,x_2)W(y_2|x_2)\\
&\times \sum_{x_3,x_4} f_2(x_2,x_3,x_4)W(y_3|x_3)W(y_4|x_4)\\
&\times \sum_{x_5} f_3(x_4,x_5)W(y_5|x_5) \qquad (22)
\end{aligned}
$$

と書き換えることができる．ここで，$W(y_1|x_1)$ に着目すると，左辺では，$x_2,\ldots,x_5$ の全てに対して総和の計算がなされているのに対し，右辺では総和の外に出されているため，積の回数が減少している．したがって，右辺に基づいて計算することによって計算量を削減できたことがわかる．

3.2 ファクターグラフ

さらに，式 (22) の各項を以下のように定義する．

$$
\begin{aligned}
M_{x_1}(x_1) &\triangleq W(y_1|x_1)M_{f_1\to x_1}(x_1)\\
M_{f_1\to x_1}(x_1) &\triangleq \sum_{x_2} f_1(x_1,x_2)M_{x_2\to f_1}(x_2)\\
M_{x_2\to f_1}(x_2) &\triangleq W(y_2|x_2)M_{f_2\to x_2}(x_2)\\
M_{f_2\to x_2}(x_2) &\triangleq \sum_{x_3,x_4} f_2(x_2,x_3,x_4)M_{x_3\to f_2}(x_3)M_{x_4\to f_2}(x_4)\\
M_{x_3\to f_2}(x_3) &\triangleq W(y_3|x_3)\\
M_{x_4\to f_2}(x_4) &\triangleq W(y_4|x_4)M_{f_3\to x_4}(x_4)\\
M_{f_3\to x_4}(x_4) &\triangleq \sum_{x_5} f_3(x_4,x_5)M_{x_5\to f_3}(x_5)\\
M_{x_5\to f_3}(x_5) &\triangleq W(y_5|x_5)
\end{aligned}
$$

これにより，式 (22) を図 2 のようなツリー構造で表すことが可能となる．

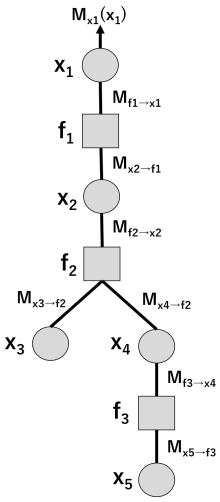

図 2　x_1 を釣り上げたツリー構造

これは，下から計算結果を持ち上げていくことにより式 (22) が求まる構造となっている．ツリー構造で表現することにより，式 (22) を視覚的に捉えることが可能となった．$x_2, \ldots, x_5$ についても同様の操作を行うと，それぞれ次のようなツリー構造を描くことができる．

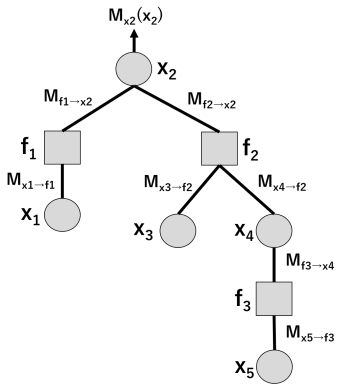

図 3　x_2 を釣り上げたツリー構造

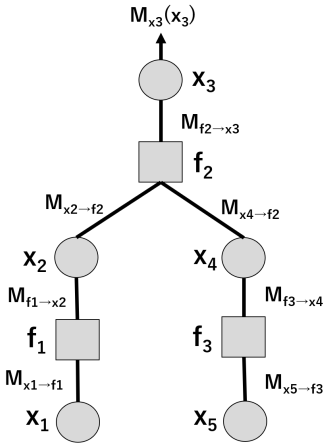

図 4　x_3 を釣り上げたツリー構造

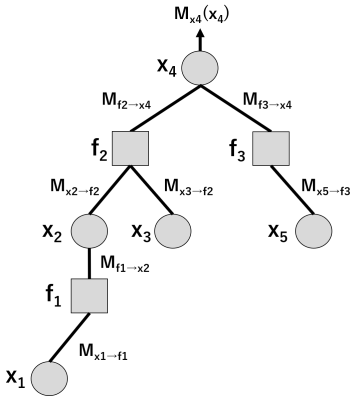

図 5　x_4 を釣り上げたツリー構造

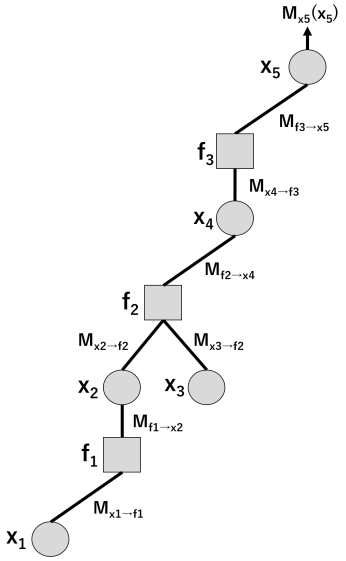

図 6　x_5 を釣り上げたツリー構造

以上で示された 5 つのツリーに従い，下から持ち上げていくように計算していくことによりそれぞれのシンボルを推定することができる．一方で，図 7 のようなファクターグラフを上下に繰り返し計算することにより，全てのシンボルを同時に推定することができる．この一連の処理によって決定したシンボルを一時推定語とし，パリティ検査行列との積を取ることで，符号語かどうかの判断が行われる．

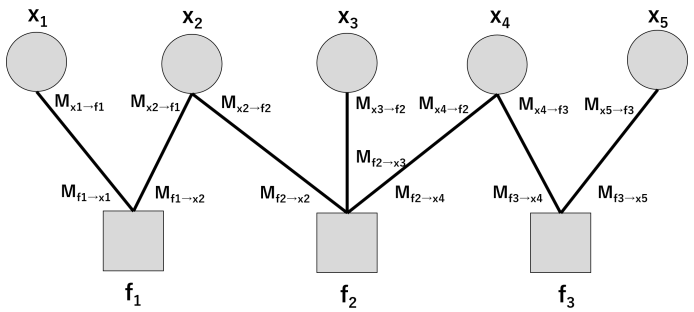

図 7　ファクターグラフ

3.3 Sum-Product アルゴリズム

続いて，以上で示された処理手順を一般化する．Sum-Product 復号は，LDPC 符号と相性の良い反復復号に代表される復号方法であり，分配則による演算量削減を系統的に行う手法をとっている．以下に，一般的な Sum-Product 復号の処理手順をステップに沿って示していく．

ステップ 1

各ノードごとでの処理を行う．変数ノードは，関数ノードにメッセージを伝達し，関数ノードは変数ノードにメッセージを伝達する．以下に，変数ノードと関数ノードにおける処理手順を計算式で示す．

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

$$M_{x_k \to f_i}(x_k) = \prod_{a \in N(x_k) \setminus f_i} M_{a \to x_k}(x_k) \tag{23}$$

として計算する．式中の $M_{x_k \to f_i}(x_k)$ は，$x_k \to f_i$ 方向に伝達されるメッセージを表しており，x_k の関数になっていることを表している．加えて，ファクターグラフに関して，求めたいメッセージのやり取りを行っているもの以外の枝の集合を $N(v)$ と表す．

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

$$M_{f_i \to x_k}(x_k) = \prod_{N(f_i) \setminus x_k} f_i(B_i) \prod_{b \in N(f_i) \setminus x_k} M_{b \to f_i}(b) \tag{24}$$

として計算する．関数ノード f_i においては，f_i ののーどから送られてきたメッセージと関数ノードの積を取る．B_i は，パリティ検査行列の i 列目において，1 が立っている行の集合を表している．

ステップ 2

それぞれのノードに入ってきたメッセージの積を計算し，周辺化関数を導出する．

周辺化関数の計算

ルートノード x_r において，

$$M_{x_r}(x_r) = \prod_{a \in N(x_r)} M_{a \to x_r}(x_r) \tag{25}$$

を計算する．
一時推定語 x_r を

$$\hat{x}_r = \arg\max M_{x_r}(x_r) \tag{26}$$

と定める．ここで得られる $\hat{x}_r$ を一時推定語と呼ぶ．

ステップ 3

ステップ 2 で求められた一時推定語にパリティ検査行列を掛け合わせる．その結果，一時推定語にパリティ検査行列を掛け合わせた積の結果が 0 になれば，一時推定語は符号語としてみなされる．仮に，積の結果が 0 にならなかった場合は，再度ステップ 1 に戻り同様の計算を繰り返す．

4　通信路の記憶を考慮した Sum-Product 復号法

本章では，3 章で示された無記憶通信路における Sum-Product 復号法をもとに，記憶のある通信路における Sum-Product 復号法の手順を示す．

4.1　尤度関数の問題点

本節では，記憶を考慮した場合における Sum-Product 復号法を前章で示された定常無記憶通信路と比較して問題点を示す．まずは通信路に記憶があるということがどういうことであるかについて示していく．パケット間隔を用いて情報を送る記憶のある通信路における時刻と時間間隔の関係を図 8 により示す．横軸は時間経過を表している．時間軸に向かっている矢印は通信路に到着した符号語の時刻を示し，時間軸から出ている矢印は通信路を通ってきた符号語を観測した時刻を示している．x_i, $d_i (i = 1, \ldots, 5)$ をパケットの時間間隔とし，$\hat{x}_i$ や $\hat{d}_i$ をパケットの到着，出発時刻とする．$\hat{x}_i$ が通信路に入っていき，サーバで処理されてから，$\hat{d}_i$ が出ていく．このとき，サーバで処理される時間があるため，$\hat{d}_1$ は遅れて通信路から出ていく．$\hat{x}_2$ に着目すると，$\hat{x}_2$ で符号語が通信路に入ってきたとき，サーバで処理されて，$\hat{d}_2$ が出てくる．しかし，$\hat{d}_2$ を考える際には，間隔 d_2 を考慮しなければならない．d_2 の間隔は $\hat{d}_1$，ひいては間隔 d_1 に依存している．パケット間隔に着目して考えた場合，x_2 という間隔が通信路に入っていく際には間隔 d_2 となって通信路を出ていく．しかし，この x_2 が通信路に入った際に d_2 がどのように変化するかは，間隔 d_1 の長さに依存していることがわかる．これが通信路に記憶があるということである．

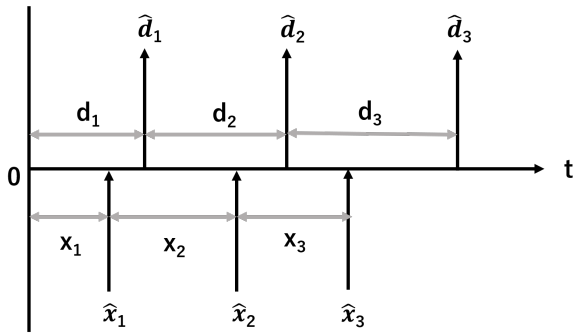

図 8　時刻と時間間隔の関係

ここで，文献 [3] で示された指数分布に基づき各シンボルに対して時間間隔を設定し，それらを足し合わせたものを $\hat{a}$ とおく．このことを踏まえて，式 (22) と同様に x_1 のシンボル MAP 復号のための尤度関数は，

$$\begin{aligned}
&\sum_{x_2,x_3,x_4,x_5} f_1(x_1,x_2)\cdot f_2(x_2,x_3,x_4)\cdot f_3(x_4,x_5)W^5(\hat{d}_1,\ldots,\hat{d}_5|\hat{a}_1,\ldots,\hat{a}_5)\\
&= W(\hat{d}_1|\hat{a}_1(x_1))\\
&\times \sum_{x_2} f_1(x_1,x_2)W(\hat{d}_2|\hat{d}_1\hat{a}_2(x_1,x_2))\\
&\times \sum_{x_3,x_4} f_2(x_2,x_3,x_4)W(\hat{d}_3|\hat{d}_2\hat{a}_3(x_1,x_2,x_3))W(\hat{d}_4|\hat{d}_3\hat{a}_4(x_1,x_2,x_3,x_4))\\
&\times \sum_{x_5} f_3(x_4,x_5)W(\hat{d}_5|\hat{d}_4\hat{a}_5(x_1,x_2,x_3,x_4,x_5)) \qquad (27)
\end{aligned}$$

と表される．Sum-Product 復号を行うには式 (22) で示されたように，各総和内の W が一変数関数となっていなければならないが，式 (27) における W は一変数関数にはなっていない．例として，三つ目の総和は無記憶通信路においては，x_4 の一変数関数としておくことが出来ていたのに対し，式 (27) においては x_4 以外の符号語が W 内に残っており，四変数関数となってしまっている．このことから，対応するツリー構造を描くことが出来ないため，従来の Sum-Product 復号法をそのまま利用することが出来ない．そこで，次節に解決策を示す．

4.2　提案法

式 (27) では，ツリー構造を描くことが出来ず，従来の Sum-Product 復号を用いて復号することが出来ないことが分かった．それに対して，文献 [3] は総和を先に計算した後，総和どうしの掛算を行うような式に変換することでツリー構造を描くことを提案した．以下に変換した式を示す．

$$
\begin{aligned}
&W(\hat{d}_1|\hat{a}_1(x_1)) \\
&\times \sum_{x_2} f_1(x_1,x_2) W(\hat{d}_2|\hat{d}_1\hat{a}_2(\hat{x}_1,x_2)) \\
&\times \sum_{x_3,x_4} f_2(x_2,x_3,x_4) W(\hat{d}_3|\hat{d}_2\hat{a}_3(\hat{x}_1,\hat{x}_2,x_3)) W(\hat{d}_4|\hat{d}_3\hat{a}_4(\hat{x}_1,\hat{x}_2,\hat{x}_3,x_4)) \\
&\times \sum_{x_5} f_3(x_4,x_5) W(\hat{d}_5|\hat{d}_4\hat{a}_5(\hat{x}_1,\hat{x}_2,\hat{x}_3,\hat{x}_4,x_5))
\end{aligned} \tag{28}
$$

ここでは，各総和において $x_1,\ldots,x_5$ を異なる変数 $\hat{x}_1,\ldots,\hat{x}_5$ に置き換えた．これにより，各総和が一変数関数となり，ツリー構造を描くための条件が成り立つ．

無記憶通信路における Sum-Prodcut 復号法と同様に，式 (28) の各項を以下のように定義する．

$$
\begin{aligned}
M_{x_1}(x_1) &\triangleq W(\hat{d}_1|\hat{a}_1(x_1)) M_{f_1\to x_1}(x_1) \\
M_{f_1\to x_1}(x_1) &\triangleq \sum_{x_2} f_1(x_1,x_2) \cdot M_{x_2\to f_1}(x_2) \\
M_{x_2\to f_1}(x_2) &\triangleq W(\hat{d}_2|\hat{d}_1\hat{a}_2(\hat{x}_1,x_2)) M_{f_2\to x_2}(x_2) \\
M_{f_2\to x_2}(x_2) &\triangleq \sum_{x_3,x_4} f_2(x_2,x_3,x_4) M_{x_3\to f_2}(x_3) M_{x_4\to f_2}(x_4) \\
M_{x_3\to f_2}(x_3) &\triangleq W(\hat{d}_3|\hat{d}_2\hat{a}_3(\hat{x}_1,\hat{x}_2,x_3)) \\
M_{x_4\to f_2}(x_4) &\triangleq W(\hat{d}_4|\hat{d}_3\hat{a}_4(\hat{x}_1,\hat{x}_2,\hat{x}_3,x_4)) M_{f_3\to x_4}(x_4) \\
M_{f_3\to x_4}(x_4) &\triangleq \sum_{x_5} f_3(x_4,x_5) M_{x_5\to f_3}(x_5) \\
M_{x_5\to f_3}(x_5) &\triangleq W(\hat{d}_5|\hat{d}_4\hat{a}_5(\hat{x}_1,\hat{x}_2,\hat{x}_3,\hat{x}_4,x_5))
\end{aligned}
$$

これにより，ツリー構造を描くことが出来るようになるため，Sum-Product 復号法の手順に基づき復号することが可能となった．文献 [3] では置き換える変数として『ひとつ前の反復処理でシンボル MAP 推定されたシンボル』を利用することが提案されている．

5　復号性能の検証

本研究では，復号誤り確率と Sum-Product 復号の復号までにかかった回数を比較することにより復号性能を検証するために 3 つの復号方法を比較する．

第 1 の方法では，『 Sum-Product 復号せずシンボル MAP 復号をした場合』について考える．記憶のある通信路において Sum-Prodcut 復号をするために式を書き換えたため，本来計算すべき式ではなくなった．そこで，Sum-Product 復号を用いた場合の復号性能を比較し検証する．第 2 の方法では，『 Sum-Product 復号法を用いた場合』について考える．また，Sum-Product 復号法を用いた場合の復号誤り確率と復号するまでにかかるループの回数を検証する．第 3 の方法では，『方法 2 における Sum-Product 復号の比較アルゴリズムを用いた場合』について考える．復号アルゴリズムにおいて，復号のタイミングを変えることにより復号誤り確率や復号するまでにかかるループの回数がどれだけ変化するのか検証する．

5.1　復号誤り確率

まず，方法 1 と方法 2 における復号誤り確率を比較したものを図 9 に示す．方法 1 と比べて方法 2 の復号誤り確率が高くなっていることがわかる．方法 2 は本来の式とは異なるものを計算しているため，方法 1 より復号性能が低下していることは自明である．また，方法 2 の復号誤り確率が 0.4 を超えてしまっているが，符号のサイズが非常に小さく，最適なものではないということもあり，この数値から方法 2 が使えない復号アルゴリズムであると判断することはできない．

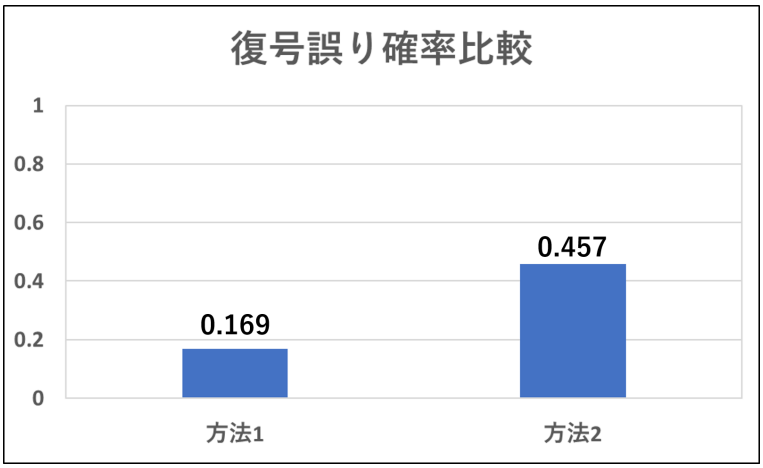

図 9　方法 1,2 の復号誤り確率比較

5.2 回数比較

本研究の処理手順は通常図 10 のように考える．図 10 では 2 回目の反復処理において必要となるシンボルは 1 回目で推定されたシンボル $x_1, \ldots, x_5$ を用いて，2 回目の反復復号を行っている．しかし，2 回目の反復処理の際に必要な一時推定語は x_1 のみである．その他のシンボルに関しては，2 回目の反復処理で一時推定語となったシンボルがそのまま用いられている．ここで，比較アルゴリズムとして図 11 を考える．このアルゴリズムでは，2 回目以降の反復処理の際に，その回で推定されたシンボルを用いて x_1 から順に復号する方法である．例えば，2 回目の反復処理における x_2 を推定する際には，2 回目の反復処理における x_1 が用いられる．

以上のことを踏まえて，復号誤り確率及び復号までにかかったループの回数の比較を行う．復号を繰り返して推定値を更新していく際に初めて条件に合致したものを符号語とし，元の符号語と比較して異なっていた場合誤りとする．すなわち，復号誤り確率が低ければ，性能の良い復号器ということが出来る．しかし，この結果からは推定値が定まるまでにかかるループの回数が復号誤り確率にどの程度影響を及ぼしているかは分からない．そこで，復号誤り確率の結果を踏まえたうえで方法 2，方法 3 において符号語が初めて現れるまでにかかった回数のグラフについて考える．

方法 2 と方法 3 における復号誤り確率を比較したものを図 12 に示す．方法 2 と比較すると復号誤り確率は低くなっており，復号性能が良くなっていることが見て取れる．しかしながら，図 13 より方法 2 の方が 1 回目でシンボルが決定する場合が多いことが分かる．これにより方法 3 の場合の方が復号するまでにかかった回数が多かったために確率が低くなった可能性も考えられる．すなわち，復号までにかかる回数が復号誤り確率に影響を及ぼしているのではないかという仮説を立てることができる．

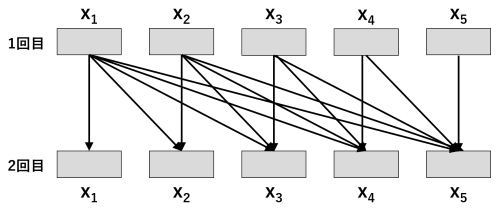

図 10　従来方法を用いた場合

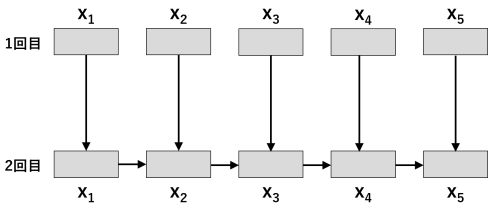

図 11　比較アルゴリズムを用いた場合

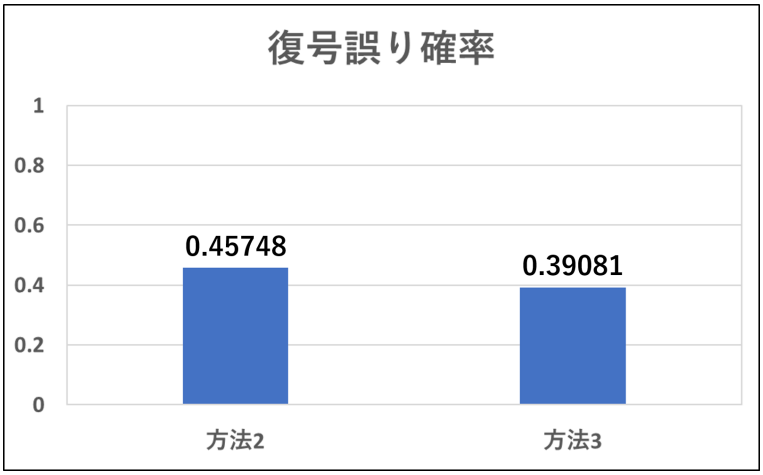

図 12　方法 2,3 の復号誤り確率比較

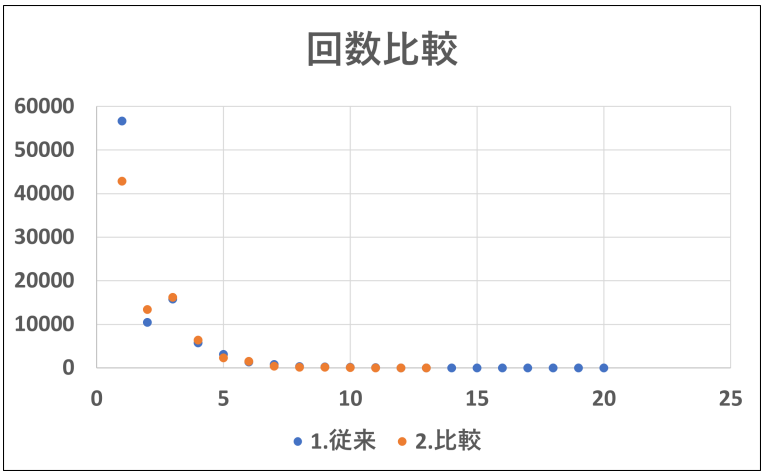

図 13　2,3 の復号までにかかったループの回数の分布図

この結果を受け方法 2，方法 3 の場合に関して復号の際に符号語の条件が成り立ってもループを止めることなく，最終的に現れた結果を符号語とみなし，改めて復号誤り確率の検証を行ったものを図 14 に示す.

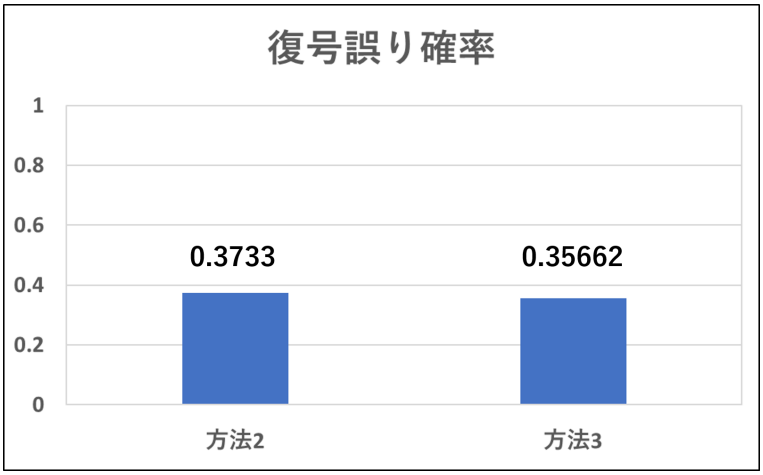

図 14　方法 2,3 の復号誤り確率比較

初めて条件に合致したものを符号語とした場合と比べて，方法 2 と方法 3 の復号誤り確率の差が小さくなったことから『方法 3 の場合の方が復号するまでにかかった回数が多かったために確率が低くなった』という可能性が高まった．復号結果が収束せず，振動し，復号回数によって誤りになる場合もならない場合もあるため復号誤り確率が同じ値に収束するとは限らない．そのため，値が振動する場合の割合も検証する必要があると考えられる．また，初めて条件に合致したものを符号語とした場合高い性能が得られると考えられていたが，値が収束した場合の方が誤り訂正確率が低くなることが分かった．

6　まとめ

本研究では，記憶のある通信路における Sum-Prodcut 復号の具体的な手法を示した．復号誤り確率の検証では，Sum-Product 復号法を用いて本来計算すべき式とは異なるものにしたことにより，シンボル MAP 復号をした場合と比べて復号性能が悪くなっている様子が確認できた．さらに，復号までにかかったループの回数比較では，初めて条件に合致したものを符号語とした場合の方が高い復号性能が得られると考えられていたが，値が収束した場合の方が復号誤り確率が低くなるということが分かった．今後は，符号のサイズを大きくするような符号器を設計し，改めて提案手法の復号性能を検証していく．

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] 市川謙吾,「パケット間隔で情報を送る通信路の記憶を考慮した Sum-Product 復号法の提案」，信州大学大学院総合理工学研究科修士論文（指導教員：西新幹彦），2020 年 3 月．

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

[5] 和田山正,「誤り訂正技術の基礎」, 森北出版,2010 年 7 月．

付録 A　ソースコード

A.1　記憶を考慮したうえでの送受信シミュレーションプログラム

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

#define SRCRATE (1/2.7182818284590452353602874713527)//レート
#define ALPHABETSIZE 4//シンボル数
#define SERVICERATE 1.0//サービスレート
#define TRIAL 100000//試行回数

#define C 5//符号語長

#define N 624
#define M 397
#define MATRIX_A 0x9908b0dfUL   /* constant vector a */
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */

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

/* 乱数 */
void init_genrand(unsigned long s)
{
    mt[0] = s & 0xffffffffUL;
    for (mti = 1; mti < N; mti++) {
        mt[mti] =
        (1812433253UL * (mt[mti-1] ^ (mt[mti-1] >> 30)) + mti);
        mt[mti] &= 0xffffffffUL;
        /* for >32 bit machines */
    }
}

void init_by_array(unsigned long init_key[], int key_length)
{
    int i, j, k;
    init_genrand(19650218UL);
    i = 1; j = 0;
    k = (N > key_length ? N : key_length);
    for (; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525UL))
          + init_key[j] + j; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++; j++;
        if (i >= N) { mt[0] = mt[N-1]; i = 1; }
        if (j >= key_length) j = 0;
    }
    for (k = N-1; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941UL))
          - i; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++;
        if (i >= N) { mt[0] = mt[N-1]; i = 1; }
    }

    mt[0] = 0x80000000UL; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0,0xffffffff]-interval */
unsigned long genrand_int32(void)
{
```

```
    unsigned long y;
    static unsigned long mag01[2] = {0x0UL, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    if (mti >= N) { /* generate N words at one time */
        int kk;

        if (mti == N+1)   /* if init_genrand() has not been called, */
            init_genrand(5489UL); /* a default initial seed is used */

        for (kk = 0;kk < N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk + 1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        for (;kk < N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1UL];

        mti = 0;
    }

    y = mt[mti++];

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return y;
}
/* generates a random number on [0,1)-real-interval */
double genrand_real2(void)
{
    return genrand_int32()*(1.0/4294967296.0);
    /* divided by 2^32 */
}

//確率変数 U
double rvariable(int x)
{
    return ((2 * x + 1) * 0.125);
}

//分布関数の逆関数
double expdistinv(double x, double rate)
{
    return(-log(1 - x) / rate);
}

//間隔変換
double sym2interval(int sym)
{
    return(expdistinv((sym + 0.5) / ALPHABETSIZE, SRCRATE));
}

//指数分布の分布関数
double expdistfunc(double x, double rate)
{
    return(1 - exp(-x * rate));
}

double max(double a, double b) {
    if (a > b)
        return a;
    else
```

```
        return b;
}

//受信時刻は送信時刻より後になる
//w(受信時刻, 受信時刻のひとつ前, 送信時刻)
double w(double dep, double pdep, double arr){
    double servicetime;
    //送信時刻が受信時刻より遅いことはあり得ない
    if(dep < arr){
        //0 を返すのはよくないかも
        return (0.0);
        }

    servicetime = dep - max(pdep,arr);
    return(exp(-servicetime));
}


/*符号器*/
//x2 と x4 の符号を乱数で作る（今回の設定では x2 と x4 が決まれば x1,x3,x5 も決まる）
void hugougo(int x[]){
    int i;
    int b[2] ;//x2 と x4 の符号
    //x2 と x4 の符号を乱数で決める
    for (int i=0 ; i < 2;i++){
        b[i] = genrand_real2() * ALPHABETSIZE;
        //printf("%f\n",genrand_real2());
        //printf("%d\n",b[i]);
    }

    //x1〜x5 のシンボルをそれぞれ求める
    //x2 と x4 は乱数で決定する 制約によりその他の符号は自動的に求まる
    //int x[5] ;
    x[1] = b[0];//x2
    x[3] = b[1];//x4

    x[0] = b[0];//x1
    x[2] = b[0] ^ b[1];//x3
    x[4] = b[1];//x5
    //printf("%d,%d,%d,%d,%d\n",x[0],x[1],x[2],x[3],x[4]);
    return;
}


/*符号器 → 通信路*/
double hugougo2interval(int x[],double a[]){
    //確率変数 U に変換
    double u[5] ;//確率変数 U
    for (int k = 0;k < 5; k++){
        a[k] = expdistinv(rvariable(x[k]), SRCRATE);//U に変換⇒間隔に変換
        //printf("%f\n",a[k]);
    }
}


/* 通信路（送受信処理） */
void transmit(double a[],double rcvtime[6])
{
    /*初期設定*/
    int i = 0;
    int j = 1;//0 を入れるため

    int k;
    double current_time = 0.0;//1. 現在時刻
    int customer_num = 0;//2. システム内のパケット数
    double sendtime[C];//3. 次の到着予定時刻 この配列を作るために

    /*到着レート*/
    //printf("%f,%f,%f,%f,%f\n",rand_lambda[0],rand_lambda[1],rand_lambda[2],rand_lambda[3],rand_lambda[4]);
```

```
    double service_end = 0.0;//4. サービス終了時刻
    for (k = 0; k < C; k++){
        sendtime[k] = 0.0;//到着予定時刻を初期化

    }
    rcvtime[0] = 0.0;

    //3. 到着予定時刻（先に決めておく）
    sendtime[0] = a[0];//arrival の a
    for (k = 1; k < C; k++){
        sendtime[k] = sendtime[k - 1] + a[k];
        //printf("%f  %f\n",rand_lambda[k],sendtime[k]);
    }

    while (i < C || customer_num > 0){
        //パケットがシステム内に入るか出ていくかの判断
        if (customer_num == 0 || (i < C && sendtime[i] < service_end)){
        //パケットがシステムに入ってくる場合
            current_time = sendtime[i];
            i++;

            //システム内にパケットがない場合
            if (customer_num == 0){
            //システム内にパケットがない場合
                service_end = current_time +   expdistinv(genrand_real2(), SERVICERATE);//サ ー ビ ス レ ー
ト rand_mu 疑似乱数を発生させる
            }
            //システム中のパケット数を 1 増やす
            customer_num++;
        }

        //パケットがシステムから出ていく場合
        else {
            current_time = service_end;
            rcvtime[j] = service_end;
            j++;
            customer_num--;

            //システム内にパケットがある場合
            if (customer_num >= 1){
                service_end = current_time + expdistinv(genrand_real2(), SERVICERATE);
            }
        }
        //printf("i:%d  current_time:%f  packet_num:%d  sendtime:%f  service_end:%f\n",i,current_time,customer_num,sendtim
    }
    return;
}


/*復号器*/
void decode1(double dep[],int xh[],int *n, int p[]){
    //printf("%d\n",x[2]);
    double ah[6]; // 復号シンボルに基づく送信時刻

    // 仮の復号シンボルと送信時刻を求める
    ah[0] = 0;
    for (int i = 0; i < 5; i++){
        //受信したパケットの『時間間隔』から指数関数を用いて『仮の復号シンボル』になおす（市川修士 p12）
        xh[i] = expdistfunc(dep[i + 1] - dep[i], SRCRATE) * ALPHABETSIZE;

        //仮の復号シンボルに基づく送信時刻を導く
        ah[i + 1] = ah[i] + sym2interval(xh[i]);
        //printf("%d %f\n", xh[i],ah[i + 1]);
        p[i] = xh[i];
    }

    double Mx1f1[4];
    double Mx2f1[4];
```

```
double Mx2f2[4];
double Mx3f2[4];
double Mx4f2[4];
double Mx4f3[4];
double Mx5f3[4];

double Mf1x1[4] = {1,1,1,1};
double Mf1x2[4] = {1,1,1,1};
double Mf2x2[4] = {1,1,1,1};
double Mf2x3[4] = {1,1,1,1};
double Mf2x4[4] = {1,1,1,1};
double Mf3x4[4] = {1,1,1,1};
double Mf3x5[4] = {1,1,1,1};

int z;
for (z = 1; z < 1000; z++){

    //○→□
    //Mx1f1[x1]=w(dh1|0a1)
    for (int x1 = 0; x1 < 4; x1++){
        Mx1f1[x1] = w(dep[1], 0, ah[0] + sym2interval(x1));
        //printf("%f\n",Mx1f1[x1]);
    }

    //Mx2f1[x2]=w(dh2|dh1ah2(xh1,x2))・Mf2x2(x2)
    for (int x2 = 0; x2 < 4; x2++){
        Mx2f1[x2] = w(dep[2], dep[1], ah[1] + sym2interval(x2)) * Mf2x2[x2];
        //printf("%f\n",Mx2f1[x2]);
    }

    //Mx2f2[x2]=w(dh2|dh1ah2(xh1,x2))・Mf1x2(x2)
    for (int x2 = 0; x2 < 4; x2++){
        Mx2f2[x2] = w(dep[2], dep[1], ah[1] + sym2interval(x2)) * Mf1x2[x2];
        //printf("%f\n",Mx2f2[x2]);
    }

    //Mx3f2[x3]=w(dh3|dh2ah3(xh1,xh2,x3))
    for (int x3 = 0; x3 < 4; x3++){
        Mx3f2[x3] = w(dep[3], dep[2], ah[2] + sym2interval(x3));
        //printf("%f\n",Mx3f2[x3]);
    }

    //Mx4f2[x4]=w(dh4|dh3ah4(xh1,xh2,xh3,x4))・Mf3x4(x4)
    for (int x4 = 0; x4 < 4; x4++){
        Mx4f2[x4] = w(dep[4], dep[3], ah[3] + sym2interval(x4)) * Mf3x4[x4];
        //printf("%f\n",Mx4f2[x4]);
    }

    //Mx4f3[x4]=w(dh4|dh3ah4(xh1xh2xh3x4))・Mf2x4(x4)
    for (int x4 = 0; x4 < 4; x4++){
        Mx4f3[x4] = w(dep[4], dep[3], ah[3] + sym2interval(x4)) * Mf2x4[x4];
        //printf("%f\n",Mx4f3[x4]);
    }

    //Mx5f3[x5]=w(dh5|dh4ah5(xh1,xh2,xh3,xh4,x5))
    for (int x5 = 0; x5 < 4; x5++){
        Mx5f3[x5] = w(dep[5], dep[4], ah[4] + sym2interval(x5));
        //printf("%f\n",Mx5f3[x5]);
    }


    //□→○
    Mf1x1[0] = Mx2f1[0];
    Mf1x1[1] = Mx2f1[1];
    Mf1x1[2] = Mx2f1[2];
    Mf1x1[3] = Mx2f1[3];

    Mf1x2[0] = Mx1f1[0];
    Mf1x2[1] = Mx1f1[1];
```

```
Mf1x2[2] = Mx1f1[2];
Mf1x2[3] = Mx1f1[3];

Mf2x2[0] = Mx3f2[0] * Mx4f2[0] + Mx3f2[1] * Mx4f2[1] + Mx3f2[2] * Mx4f2[2] + Mx3f2[3] * Mx4f2[3];
Mf2x2[1] = Mx3f2[1] * Mx4f2[0] + Mx3f2[0] * Mx4f2[1] + Mx3f2[3] * Mx4f2[2] + Mx3f2[2] * Mx4f2[3];
Mf2x2[2] = Mx3f2[2] * Mx4f2[0] + Mx3f2[0] * Mx4f2[2] + Mx3f2[3] * Mx4f2[1] + Mx3f2[1] * Mx4f2[3];
Mf2x2[3] = Mx3f2[3] * Mx4f2[0] + Mx3f2[0] * Mx4f2[3] + Mx3f2[2] * Mx4f2[1] + Mx3f2[1] * Mx4f2[2];

Mf2x3[0] = Mx2f2[0] * Mx4f2[0] + Mx2f2[1] * Mx4f2[1] + Mx2f2[2] * Mx4f2[2] + Mx2f2[3] * Mx4f2[3];
Mf2x3[1] = Mx2f2[1] * Mx4f2[0] + Mx2f2[0] * Mx4f2[1] + Mx2f2[3] * Mx4f2[2] + Mx2f2[2] * Mx4f2[3];
Mf2x3[2] = Mx2f2[2] * Mx4f2[0] + Mx2f2[0] * Mx4f2[2] + Mx2f2[3] * Mx4f2[1] + Mx2f2[1] * Mx4f2[3];
Mf2x3[3] = Mx2f2[3] * Mx4f2[0] + Mx2f2[0] * Mx4f2[3] + Mx2f2[2] * Mx4f2[1] + Mx2f2[1] * Mx4f2[2];

Mf2x4[0] = Mx2f2[0] * Mx3f2[0] + Mx2f2[1] * Mx3f2[1] + Mx2f2[2] * Mx3f2[2] + Mx2f2[3] * Mx3f2[3];
Mf2x4[1] = Mx2f2[1] * Mx3f2[0] + Mx2f2[0] * Mx3f2[1] + Mx2f2[3] * Mx3f2[2] + Mx2f2[2] * Mx3f2[3];
Mf2x4[2] = Mx2f2[2] * Mx3f2[0] + Mx2f2[0] * Mx3f2[2] + Mx2f2[3] * Mx3f2[1] + Mx2f2[1] * Mx3f2[3];
Mf2x4[3] = Mx2f2[3] * Mx3f2[0] + Mx2f2[0] * Mx3f2[3] + Mx2f2[2] * Mx3f2[1] + Mx2f2[1] * Mx3f2[2];

Mf3x4[0] = Mx5f3[0];
Mf3x4[1] = Mx5f3[1];
Mf3x4[2] = Mx5f3[2];
Mf3x4[3] = Mx5f3[3];

Mf3x5[0] = Mx4f3[0];
Mf3x5[1] = Mx4f3[1];
Mf3x5[2] = Mx4f3[2];
Mf3x5[3] = Mx4f3[3];


//最大化
//x1
double max1 = -1.0;
for (int x1 = 0; x1 < 4; x1++){
    double lh = w(dep[1], 0, ah[0] + sym2interval(x1)) * Mf1x1[x1];
    if (max1 < lh){
        max1 = lh;
        xh[0] = x1;
    }
}
//printf("%f\n",max1);
//printf("%d\n",xh[0]);


//x2
double max2 = -1.0;
for (int x2 = 0; x2 < 4; x2++){
    double mh = w(dep[2], dep[1], ah[1] + sym2interval(x2)) * Mf1x2[x2] * Mf2x2[x2];
    if (max2 < mh){
        max2 = mh;
        xh[1] = x2;
    }
}
//printf("%f\n",max2);
//printf("%d\n",xh[1]);

//x3
double max3 = -1.0;
for (int x3 = 0; x3 < 4; x3++){
    double nh =  w(dep[3], dep[2], ah[2] + sym2interval(x3)) * Mf2x3[x3];
    if (max3 < nh){
        max3 = nh;
        xh[2] = x3;
    }
}
//printf("%f\n",max3);
//printf("%d\n",xh[2]);

double max4 = -1.0;
for (int x4 = 0; x4 < 4; x4++){
```

```
            double oh = w(dep[4], dep[3], ah[3] + sym2interval(x4)) * Mf2x4[x4] * Mf3x4[x4];
            if (max4 < oh){
                max4 = oh;
                xh[3] = x4;
            }
        }
        //printf("%f\n",max4);
        //printf("%d\n",xh[3]);

        double max5 = -1.0;
        for (int x5 = 0; x5 < 4; x5++){
            double ph = w(dep[5], dep[4], ah[4] + sym2interval(x5)) * Mf3x5[x5];
            if (max5 < ph){
                max5 = ph;
                xh[4] = x5;
            }
        }
        //printf("%f\n",max5);
        //printf("%d\n",xh[4]);


        //符号語かどうかの判定
        //xh[0],...,xh[4] が符号語だったらループを抜ける
        if (((xh[0]^xh[1]) == 0) && ((xh[1]^xh[2]^xh[3]) == 0) && ((xh[3]^xh[4]) == 0)){
           break;//ループを抜ける
        }

        for (int j = 0; j < 5; j++){
            ah[j + 1] = ah[j] + sym2interval(xh[j]);
            //printf("%d %f\n", xh[i],ah[i + 1]);
        }
    }
    //printf("%d,%d,%d,%d,%d\n",xh[0],xh[1],xh[2],xh[3],xh[4]);
    *n = z;
    //printf("%d\n",&n);

    return;
}

void decode2(double dep[],int xh[]){
    double ah[6]; //復号シンボルに基づく送信時刻
    //x1 を推定
    double max1 = -1.0;
    for (int x1 = 0; x1 < 4; x1++){
        double pra = 0;
        ah[0] = sym2interval(x1);
        for (int x2 = 0; x2 < 4; x2++){
            ah[1] = ah[0] + sym2interval(x2);
            for (int x3 = 0; x3 < 4; x3++){
                ah[2] = ah[1] + sym2interval(x3);
                for (int x4 = 0; x4 < 4; x4++){
                    ah[3] = ah[2] + sym2interval(x4);
                    for (int x5 = 0; x5 < 4; x5++){
                        ah[4] = ah[3] + sym2interval(x5);
                        if ((x1^x2) != 0 || (x2^x3^x4) != 0 || (x4^x5) != 0){
                            continue;
                        }
                    pra += w(dep[1],dep[0],ah[0])
                         * w(dep[2],dep[1],ah[1])
                         * w(dep[3],dep[2],ah[2])
                         * w(dep[4],dep[3],ah[3])
                         * w(dep[5],dep[4],ah[4]);
                    }
                }
            }
        }
        if (max1 < pra){
            max1 = pra;
            xh[0] = x1;
```

```
    }
    //printf("%d\n",xh[0]);
}


//x2 を推定
double max2 = -1.0;
for (int x2 = 0;x2 <4 ;x2++){
    double prb = 0;
    for (int x1 = 0;x1 < 4;x1++){
        ah[0] = sym2interval(x1);
        ah[1] = ah[0] + sym2interval(x2);
        for (int x3 = 0;x3<4;x3++){
            ah[2] = ah[1] + sym2interval(x3);
            for (int x4 = 0;x4<4;x4++){
                ah[3] = ah[2] + sym2interval(x4);
                for (int x5 = 0;x5 < 4;x5++){
                    ah[4] = ah[3] + sym2interval(x5);
                    if ((x1^x2) != 0 || (x2^x3^x4) != 0 || (x4^x5) != 0){
                        continue;
                    }
                    prb += w(dep[1],dep[0],ah[0])
                         * w(dep[2],dep[1],ah[1])
                         * w(dep[3],dep[2],ah[2])
                         * w(dep[4],dep[3],ah[3])
                         * w(dep[5],dep[4],ah[4]);
                }
            }
        }
    }
    if (max2 < prb){
        max2 = prb;
        xh[1] = x2;
    }
}

//x3 を推定
double max3 = -1.0;
for (int x3 = 0; x3 < 4 ;x3++){
    double prc = 0;
    for (int x1 = 0; x1 < 4; x1++){
        ah[0] = sym2interval(x1);
        for (int x2 = 0; x2 < 4; x2++){
            ah[1] = ah[0] + sym2interval(x2);
            for (int x4 = 0; x4 < 4; x4++){
                ah[2] = ah[1] + sym2interval(x3);
                ah[3] = ah[2] + sym2interval(x4);
                for (int x5 = 0;x5 < 4;x5++){
                    ah[4] = ah[3] + sym2interval(x5);
                    if ((x1^x2) != 0 || (x2^x3^x4) != 0 || (x4^x5) != 0){
                        continue;
                    }
                    prc += w(dep[1],dep[0],ah[0])
                         * w(dep[2],dep[1],ah[1])
                         * w(dep[3],dep[2],ah[2])
                         * w(dep[4],dep[3],ah[3])
                         * w(dep[5],dep[4],ah[4]);
                }
            }
        }
    }
    if (max3 < prc){
        max3 = prc;
        xh[2] = x3;
    }
}

//x4 を推定
double max4 = -1.0;
```

```
    for (int x4 = 0;x4 <4 ;x4++){
        double prd = 0;
        for (int x1 = 0;x1 < 4; x1++){
            ah[0] = sym2interval(x1);
            for (int x2 = 0; x2 < 4; x2++){
                ah[1] = ah[0] + sym2interval(x2);
                for (int x3 = 0; x3 < 4; x3++){
                    ah[2] = ah[1] + sym2interval(x3);
                    ah[3] = ah[2] + sym2interval(x4);
                    for (int x5 = 0;x5 < 4;x5++){
                        ah[4] = ah[3] + sym2interval(x5);
                        if ((x1^x2) != 0 || (x2^x3^x4) != 0 || (x4^x5) != 0){
                            continue;
                        }
                        prd += w(dep[1],dep[0],ah[0])
                             * w(dep[2],dep[1],ah[1])
                             * w(dep[3],dep[2],ah[2])
                             * w(dep[4],dep[3],ah[3])
                             * w(dep[5],dep[4],ah[4]);
                    }
                }
            }
        }
        if (max4 < prd){
            max4 = prd;
            xh[3] = x4;
        }
        //printf("%d\n",xh[3]);
    }

    //x5 を推定
    double max5 = -1.0;
    for (int x5 = 0; x5 < 4 ; x5++){
        double pre = 0;
        for (int x1 = 0; x1 < 4; x1++){
            ah[0] = sym2interval(x1);
            for (int x2 = 0; x2 < 4; x2++){
                ah[1] = ah[0] + sym2interval(x2);
                for (int x3 = 0; x3 < 4; x3++){
                    ah[2] = ah[1] + sym2interval(x3);
                    for (int x4 = 0; x4 < 4; x4++){
                        ah[3] = ah[2] + sym2interval(x4);
                        ah[4] = ah[3] + sym2interval(x5);
                        if ((x1^x2) != 0 || (x2^x3^x4) != 0 || (x4^x5) != 0){
                            continue;
                        }
                        pre += w(dep[1],dep[0],ah[0])
                             * w(dep[2],dep[1],ah[1])
                             * w(dep[3],dep[2],ah[2])
                             * w(dep[4],dep[3],ah[3])
                             * w(dep[5],dep[4],ah[4]);
                    }
                }
            }
        }
        if (max5 < pre){
            max5 = pre;
            xh[4] = x5;
        }
    }
    //printf("%d,%d,%d,%d,%d\n",xh[0],xh[1],xh[2],xh[3],xh[4]);
    return;
}


void decode3(double dep[],int xh[],int *n, int p[]){

    double ah[6]; // 復号シンボルに基づく送信時刻
    // 仮の復号シンボルと送信時刻を求める
```

```
ah[0] = 0;
for (int i = 0; i < 5; i++){
    //受信したパケットの『時間間隔』から指数関数を用いて『仮の復号シンボル』になおす（市川修士 p12）
    xh[i] = expdistfunc(dep[i + 1] - dep[i], SRCRATE) * ALPHABETSIZE;

    //仮の復号シンボルに基づく送信時刻を導く
    ah[i + 1] = ah[i] + sym2interval(xh[i]);
    //printf("%f\n",sym2interval(xh[i]));
    //printf("%f\n",ah[i+1]);
    p[i] = xh[i];
}
//printf("str:%d%d%d%d%d\n",xh[0],xh[1],xh[2],xh[3],xh[4]);

double Mx1f1[4];
double Mx2f1[4];
double Mx2f2[4];
double Mx3f2[4];
double Mx4f2[4];
double Mx4f3[4];
double Mx5f3[4];

double Mf1x1[4] = {1,1,1,1};
double Mf1x2[4] = {1,1,1,1};
double Mf2x2[4] = {1,1,1,1};
double Mf2x3[4] = {1,1,1,1};
double Mf2x4[4] = {1,1,1,1};
double Mf3x4[4] = {1,1,1,1};
double Mf3x5[4] = {1,1,1,1};

int z;
for (z = 1; z < 100; z++){
    //○→□
    //Mx1f1[x1]=w(dh1|0a1)
    for (int x1 = 0; x1 < 4; x1++){
        Mx1f1[x1] = w(dep[1], 0, ah[0] + sym2interval(x1));
        //printf("%f\n",Mx1f1[x1]);
    }


    //Mx2f1[x2]=w(dh2|dh1ah2(xh1,x2))・Mf2x2(x2)
    for (int x2 = 0; x2 < 4; x2++){
        Mx2f1[x2] = w(dep[2], dep[1], ah[1] + sym2interval(x2)) * Mf2x2[x2];
        //printf("%f\n",Mx2f1[x2]);
    }

    //Mx2f2[x2]=w(dh2|dh1ah2(xh1,x2))・Mf1x2(x2)
    for (int x2 = 0; x2 < 4; x2++){
        Mx2f2[x2] = w(dep[2], dep[1], ah[1] + sym2interval(x2)) * Mf1x2[x2];
        //printf("%f\n",Mx2f2[x2]);
    }

    //Mx3f2[x3]=w(dh3|dh2ah3(xh1,xh2,x3))
    for (int x3 = 0; x3 < 4; x3++){
        Mx3f2[x3] = w(dep[3], dep[2], ah[2] + sym2interval(x3));
        //printf("%f\n",Mx3f2[x3]);
    }

    //Mx4f2[x4]=w(dh4|dh3ah4(xh1,xh2,xh3,x4))・Mf3x4(x4)
    for (int x4 = 0; x4 < 4; x4++){
        Mx4f2[x4] = w(dep[4], dep[3], ah[3] + sym2interval(x4)) * Mf3x4[x4];
        //printf("%f\n",Mx4f2[x4]);
    }

    //Mx4f3[x4]=w(dh4|dh3ah4(xh1xh2xh3x4))・Mf2x4(x4)
    for (int x4 = 0; x4 < 4; x4++){
        Mx4f3[x4] = w(dep[4], dep[3], ah[3] + sym2interval(x4)) * Mf2x4[x4];
        //printf("%f\n",Mx4f3[x4]);
    }
```

```
//Mx5f3[x5]=w(dh5|dh4ah5(xh1,xh2,xh3,xh4,x5))
for (int x5 = 0; x5 < 4; x5++){
    Mx5f3[x5] = w(dep[5], dep[4], ah[4] + sym2interval(x5));
    //printf("%f\n",Mx5f3[x5]);
}

//□→○
Mf1x1[0] = Mx2f1[0];
Mf1x1[1] = Mx2f1[1];
Mf1x1[2] = Mx2f1[2];
Mf1x1[3] = Mx2f1[3];
//printf("%f\n",Mf1x1[0]);

Mf1x2[0] = Mx1f1[0];
Mf1x2[1] = Mx1f1[1];
Mf1x2[2] = Mx1f1[2];
Mf1x2[3] = Mx1f1[3];
//printf("%f\n",Mf1x2[0]);

Mf2x2[0] = Mx3f2[0] * Mx4f2[0] + Mx3f2[1] * Mx4f2[1] + Mx3f2[2] * Mx4f2[2] + Mx3f2[3] * Mx4f2[3];
Mf2x2[1] = Mx3f2[1] * Mx4f2[0] + Mx3f2[0] * Mx4f2[1] + Mx3f2[3] * Mx4f2[2] + Mx3f2[2] * Mx4f2[3];
Mf2x2[2] = Mx3f2[2] * Mx4f2[0] + Mx3f2[0] * Mx4f2[2] + Mx3f2[3] * Mx4f2[1] + Mx3f2[1] * Mx4f2[3];
Mf2x2[3] = Mx3f2[3] * Mx4f2[0] + Mx3f2[0] * Mx4f2[3] + Mx3f2[2] * Mx4f2[1] + Mx3f2[1] * Mx4f2[2];
//printf("%f\n",Mf2x2[0]);

Mf2x3[0] = Mx2f2[0] * Mx4f2[0] + Mx2f2[1] * Mx4f2[1] + Mx2f2[2] * Mx4f2[2] + Mx2f2[3] * Mx4f2[3];
Mf2x3[1] = Mx2f2[1] * Mx4f2[0] + Mx2f2[0] * Mx4f2[1] + Mx2f2[3] * Mx4f2[2] + Mx2f2[2] * Mx4f2[3];
Mf2x3[2] = Mx2f2[2] * Mx4f2[0] + Mx2f2[0] * Mx4f2[2] + Mx2f2[3] * Mx4f2[1] + Mx2f2[1] * Mx4f2[3];
Mf2x3[3] = Mx2f2[3] * Mx4f2[0] + Mx2f2[0] * Mx4f2[3] + Mx2f2[2] * Mx4f2[1] + Mx2f2[1] * Mx4f2[2];
//printf("%f\n",Mf2x3[0]);

Mf2x4[0] = Mx2f2[0] * Mx3f2[0] + Mx2f2[1] * Mx3f2[1] + Mx2f2[2] * Mx3f2[2] + Mx2f2[3] * Mx3f2[3];
Mf2x4[1] = Mx2f2[1] * Mx3f2[0] + Mx2f2[0] * Mx3f2[1] + Mx2f2[3] * Mx3f2[2] + Mx2f2[2] * Mx3f2[3];
Mf2x4[2] = Mx2f2[2] * Mx3f2[0] + Mx2f2[0] * Mx3f2[2] + Mx2f2[3] * Mx3f2[1] + Mx2f2[1] * Mx3f2[3];
Mf2x4[3] = Mx2f2[3] * Mx3f2[0] + Mx2f2[0] * Mx3f2[3] + Mx2f2[2] * Mx3f2[1] + Mx2f2[1] * Mx3f2[2];
//printf("%f\n",Mf2x4[0]);

Mf3x4[0] = Mx5f3[0];
Mf3x4[1] = Mx5f3[1];
Mf3x4[2] = Mx5f3[2];
Mf3x4[3] = Mx5f3[3];
//printf("%f\n",Mf3x4[0]);

Mf3x5[0] = Mx4f3[0];
Mf3x5[1] = Mx4f3[1];
Mf3x5[2] = Mx4f3[2];
Mf3x5[3] = Mx4f3[3];
//printf("%f\n",Mf3x5[0]);


//最大化
//x1
double max1 = -1.0;
for (int x1 = 0;x1 < 4;x1++){
    double lh = w(dep[1], 0, ah[0] + sym2interval(x1)) * Mf1x1[x1];
    if (max1 < lh){
        max1 = lh;
        xh[0] = x1;
    }
}
ah[1] = ah[0] + sym2interval(xh[0]);
//printf("%d\n",xh[0]);

//x2
double max2 = -1.0;
for (int x2 = 0;x2 < 4;x2++){
    double mh = w(dep[2], dep[1], ah[1] + sym2interval(x2)) * Mf1x2[x2] * Mf2x2[x2];
    if (max2 <  mh){
        max2 =  mh;
```

```
                xh[1] = x2;
                }
        }
        ah[2] = ah[1] + sym2interval(xh[1]);
        //printf("%d\n",xh[1]);


        //x3
        double max3 = -1.0;
        for (int x3 = 0;x3 < 4;x3++){
            double nh =  w(dep[3], dep[2], ah[2] + sym2interval(x3)) * Mf2x3[x3];
            if (max3 < nh){
                max3 = nh;
                xh[2] = x3;
            }
        }
        ah[3] = ah[2] + sym2interval(xh[2]);
        //printf("%d\n",xh[2]);


        double max4 = -1.0;
        for (int x4 = 0;x4 < 4;x4++){
            double oh = w(dep[4], dep[3], ah[3] + sym2interval(x4)) * Mf2x4[x4] * Mf3x4[x4];
            if (max4 < oh){
                max4 = oh;
                xh[3] = x4;
            }
        }
        ah[4] = ah[3] + sym2interval(xh[3]);
        //printf("%d\n",xh[3]);

        double max5 = -1.0;
        for (int x5 = 0;x5 < 4;x5++){
            double ph = w(dep[5], dep[4], ah[4] + sym2interval(x5)) * Mf3x5[x5];
            if (max5 < ph){
                max5 = ph;
                xh[4] = x5;
            }
        }
        //ah[5] = ah[4] + sym2interval(xh[4]);//更新する必要ない
        //printf("%d\n",xh[4]);

        //printf("end:%d%d%d%d%d\n",xh[0],xh[1],xh[2],xh[3],xh[4]);
        //printf("***********\n");

        //符号語かどうかの確認
        //xh[0],...,xh[4] が符合語だったらループを抜ける
        if (((xh[0]^xh[1]) == 0) && ((xh[1]^xh[2]^xh[3]) == 0) && ((xh[3]^xh[4]) == 0)){
            break;//ループを抜ける
        }
    }
    //printf("%d,%d,%d,%d,%d\n",xh[0],xh[1],xh[2],xh[3],xh[4]);
    //printf("%d\n",n);
    *n = z;
    return;
}


int main(){
    FILE *fp;
    unsigned long init[4] = {0x123, 0x234, 0x345, 0x456};
    init_by_array(init, 4);

    int x[5];//元のシンボル
    int xh[5];//仮の復号シンボル
    double a[5];//間隔
    double d[6];//時刻
    int n = 0;
```

```
    int p[5];

    fp = fopen("decode1.txt", "w");//ファイルに書き込む

    int error = 0;//ループ内での誤り数
    for(int j = 0;j < TRIAL ;j++){
        int count1 = 0;
        int count = 0;

        hugougo(x);//元のシンボルを配列 x に入れる
        hugougo2interval(x,a);//元の間隔を配列 a に入れる
        transmit(a,d);//間隔を配列 d に入れる

        //decode1(d,xh,&n,p);
        //decode2(d,xh);
        decode3(d,xh,&n,p);

        //printf("%d\n",n);
        //fprintf(fp, "%d\n",n);

        for (count = 0;count < 5;count++){
            if(xh[count] != x[count]){
                break;
            }
        }

        //誤りがある場合エラー数をカウントする
        //復号した値が符号語となるか判定
        if (count < 5){
            error++;
            //fprintf(fp, "0\n");//最終的にエラーだったら 0 と表示
            //fprintf(fp, "%d\n",n);
            //fprintf(fp, "100\n");
        }
        else{
            //fprintf(fp, "1\n");//最終的に正しかったら 1 と表示
            //fprintf(fp, "%d\n",n);
            //fprintf(fp, "100\n");
        }
    }
    printf("%f\n", (double)error/TRIAL);
    fclose(fp);
}
```