信州大学大学院総合理工学研究科

修士論文

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 復号法の提案

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　18W2013A
氏名　市川　謙吾

2020 年 2 月 24 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ー通信において，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使用することができる．また通信経路の選択をすることができるのでネットワーク上で一部に障害が生じてしまったとしてもほかの回線を利用し通信を行うことができ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が，Anantharam and Verdú[1] によると, パケット間隔も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．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になる．

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，復号器を作製しており，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] で示されている内容は，通信路の記憶の有無や符号シンボルの数などの様々な制約が存在している．しかし，この制約が結果にどれほど影響を与えているかは分かっていない．文献 [3] で示されている従来研究では，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，通信路行列を算出している．そして，算出された通信路行列から通信路容量を求め，文献 [2] で達成された符号化レートと比較している．そして，本研究では，この従来研究で通信路に記憶のない状態での sum-product 復号について考察されている文献 [3] に派生する内容として，実際の通信路には記憶があるという事実に基づいて，どのように復号されているかについて考察する．本研究では最初に，シンボル数，符号語の種類を定める．復号に関しては，復号器で観測されたパケット間隔に基づいて，符号語を sum-product 復号法によって復号する．そして，前提として sum-product 復号をするうえで，ツリー構造

を描くことができ，通常の処理で復号が可能な復号方法を考えている．その中で，復号するうえでの問題点が発生した．その問題とは，前提であるツリー構造を書ける sum-product 復号ができないという点である．そこで，本研究では前提に基づいた sum-product 復号を行うために，問題点に対しての解決策を提示する．

1.2　本論文の構成

本論文は次のような章から構成されている．2 章では本研究で扱う通信路モデル，誤り訂正符号などの通信システムについて述べる．3 章では LDPC 符号の詳細，パリティ検査行列による符号化などの誤り訂正符号の詳細に加えて，復号法の詳細についても述べる．4 章では本研究において復号する際に問題になった箇所とその解決方法についての詳細を述べる．最後に 5 章で結果と考察に加え, 本研究で実装した改良版アルゴリズムについても述べる.

2　通信システム

本章では本研究で扱う通信システムを説明する．説明の多くは従来研究 [3] からの引用である．

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．そして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を設置する．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．

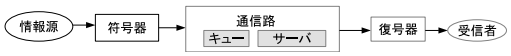

図 1　通信路モデル

さらに，パケットの中に情報を入れて通信するのが普通のパケット通信であるが，本研究では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．符号器と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実数の並びであり，本研究では文献 [1] に従い，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 $(a_1, a_2, \cdots, a_n)$ として表される．符号器が符号語 $(a_1, a_2, \cdots, a_n)$ を送信するとき, 時刻 0 で空のキューに対して時刻 $\sum_{i=1}^{k} a_i$ 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．

2.3 誤り訂正符号の概要

送信者から受信者に情報を送る際に，通信路にはノイズが発生してしまい，受信側では送信者と完全に同じ時間間隔を受け取ることができない．そこで送信者が送った情報と同じ情報を受け取ることができるように，受信者側では誤り訂正技術が必要である．誤り訂正の基本的な設定のもとで，Shannon は理論的な限界値である通信路容量を示した．Shannon が示した通信路符号化定理の証明は，ランダムに構成された符号の中に，この限界を達成することができるものが存在するという手法を用いている．しかし，ランダムに構成した符号を実際の符号化に用いることは現実的ではない．従来研究 [1] によって，ランダム符号化ではパケット間隔を指数分布で発生させると，理論的には通信路容量を達成できるという結果が導かれている．本研究ではランダム符号化は用いないが，従来研究の結果から指数分布によってパケット間隔と符号シンボルを関連付ける．シンボルは 4 種類として，一様分布とする．

そして，sum-product 復号に対する符号化方法として，本研究では近年シャノン限界値に近い値を出せたことで注目されている LDPC 符号化を用いる．LDPC 符号は，情報伝送レートの理論上の上限値である通信路容量に極めて近いレートを達成した最初の符号であり，今日においても最も効率の良い符号である．なお，文献 [2] の中でも，符号化は LDPC 符号を用いている．

3　誤り訂正符号の一般論

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って定義される線形符号である．本研究ではパリティ検査行列の各行，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．この章ではこの LDPC 符号の詳細，そして復号方法についても触れていく．

3.1　パリティ検査行列に基づく符号化

本研究で扱う符号である LDPC 符号は, パリティ検査行列によって符号語が決定される．本研究では 2 元 m 行 n 列の疎なパリティ検査行列 H を考える．このパリティ検査行列は，作成者によって自由に設定する事は可能であるが，すべての行の値は違わないといけない，すべてが 0 の列があってはならないなどの制約もある．すべての行の値は違わないといけない制約に関しては，仮に同じ行の値が存在してしまうとファクターグラフのエッジが重なってしまうためである．本研究においては，2 元 m 行 n 列 $(0 < m < n)$ の疎なパリティ検査行列を考える．そして，2 元線形符号 C が

$$C \triangleq \{c \in \mathbb{F}_2^n | cH^T = 0\} \tag{1}$$

と定義される．H が疎行列 (行列中の 1 の個数が非常に少ない) の場合，C は LDPC 符号と呼ばれる．H^T は行列の転置を表しており，$\mathbb{F}_2$ は 2 元線形符号であることを表している．符号語は $cH^T = 0$ を満たすことから，連立方程式を解くことによって行う．例として，

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = 0 \tag{2}$$

のようなパリティ検査行列を用いた連立方程式などがあげられる．式 (2) で表した連立方程式が成り立つと，$a_1 \cdots a_5$ のシンボルは符号語としてみなされる．式 (2) の行列は，先ほど述べたように作成者が制約の範囲内で自由に設計することが可能である．

そして，この検査行列を用いてファクターグラフを書くことが可能である．ファクターグラフとは，n 変数関数からなる多変数関数 $f(x_1.\ x_2 \cdots x_n)$ の因子分解を表している．なおかつ，検査行列から導くことができる 2 部グラフであり，LDPC 符号の性質を議論する 1 つの

ものさしとなる．ファクターグラフの上部は変数ノード，下部は関数ノードと呼ばれ，それぞれパリティ検査行列の行と列に対応している．そして，ファクターグラフにおいて枝が，それぞれの変数ノードと関数ノード間でのメッセージのやり取りの流れを表しており，端から端までメッセージが伝達される様子を表している．この枝は，パリティ検査行列の中の 1 と対応している．検査行列の中で 1 がある位置に対応している，変数ノードと関数ノードを枝で結ぶことでファクターグラフが構成される．そして図 2 に示されているようなファクターグラフだけではなく，各ノード間でループが入るようなファクターグラフを描けるような，検査行列も設計することができる．このように、検査行列の設計者によって様々なファクターグラフを作ることができる．

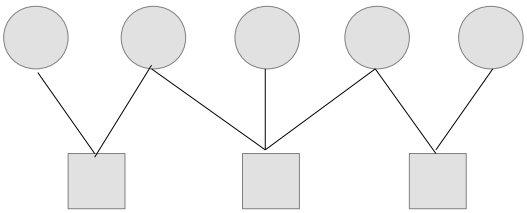

図 2　ファクターグラフ

3.2　復号法の詳細

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る．この sum-product アルゴリズムは，分配則による演算量削減を系統的に行う手法であり，一般的な sum-product 復号のおおよその処理の流れを示す．

ステップ 1

パリティ検査行列を用いて，ファクターグラフを構成する．符号器から送られてきた正しい符号シンボルを求めたい変数ノードを根として木の形となるように変数ノードを配置する．なお，この構造をツリー構造と言い，sum-product 復号の計算の流れを表している．例として，図 2 の上部ノードである変数ノードの中で，左端の変数を求める際のツリー構造を図 3 に示す．ツリー構造を書くには，求めたい変数の位置を手に持って釣り上げた時にできる構造を考えると書くことができる．

ステップ 2

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．このメッセージ時のやり取りはファクターグラフを用

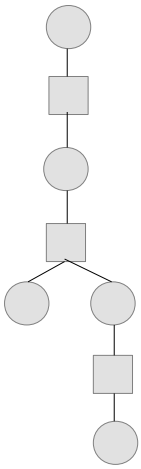

図 3　ツリー構造

いて行われ，ツリー構造を見ることで流れの順序を確認することができる．以下に，変数ノードと関数ノードの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

$$M_{x_k \to f_i}(x_k) = \prod_{a \in N(x_k) \setminus f_i} M_{a \to x_k}(x_k) \tag{3}$$

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．そして，ファクターグラフにいて，求めたいメッセージのやり取りを行っている枝以外の枝の集合を $N_{(v)}$ と表す．

関数ノードの処理

関数ノード f_i から変数ノード x_k へのメッセージを

$$M_{f_i \to x_k}(x_k) = \sum_{N(f_i) \setminus x_k} f_i(B_i) \prod_{b \in N(f_i) \setminus x_k} M_{b \to f_i}(b) \tag{4}$$

として計算する．関数ノード f_i においては，f_i のノード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 3

それぞれのノードに入ってきたメッセージの積を計算する．そして送られてきたメッセージで周辺化を行い，そのシンボルを一時推定語とする．

周辺化関数の計算ルートノード x_r において，

$$M_{x_r}(x_r) = \prod_{a \in N(x_r)} M_{a \to x_r}(x_r) \tag{5}$$

を計算する．

ステップ 4

ステップ 3 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，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 2 に戻り同様の計算を繰り返す．

次に，本研究においての設定を踏まえながら，sum-product 復号の一連の計算方法について具体的な例を用いて説明する．本研究では，$(d_1, d_2, \cdots d_5)$ の 5 つのパケット間隔である時間間隔を観測し，元の符号語 $(x_1, x_2, \cdots x_5)$ の 5 つの符号シンボルを復号する．そして関数ノードを (f_1, f_2, f_3) の 3 つ，そして本研究における最後制約としてシンボルを 0，1，2，3 の 4 種類と固定する．なお，パリティ検査行列は，式 (2) で示した行列を用いる．これらの条件を加味したうえで，ファクターグラフを図 4 のように書くことができる．

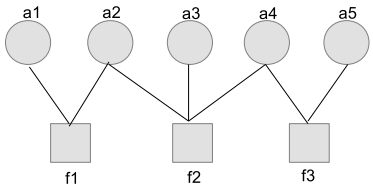

図 4　本研究におけるファクターグラフ

以上の手順で処理が行われている．次に無記憶通信路での一般的な sum-product 復号の手順として図 4 の a_1 の一時推定語を求める操作を説明していく．

まず，a_1 を求める際に，ツリー構造を考える必要がある．このツリー構造を図 5 に示す．先ほど説明したように，ツリー構造は，下のノードから計算していく様子を分かりやすく可視化したものである．

一時推定語を求めるための式は，

$$P_{\bar{X}|\bar{D}}(\bar{x}|\bar{d}) = \frac{P_{\bar{X}}(\bar{x}) P_{\bar{D}|\bar{X}}(\bar{d}|\bar{x})}{P_{\bar{D}}(\bar{d})} \tag{6}$$

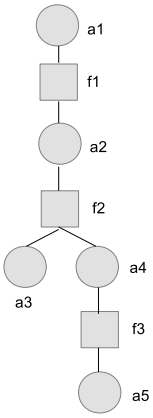

図 5　a_1 を釣り上げたツリー構造

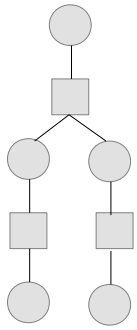

図 6　a_2 を釣り上げたツリー構造

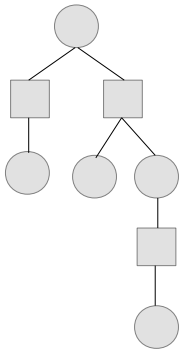

図 7　a_3 を釣り上げたツリー構造

によって表される．この式では，符号語を $X = (x_1, x_2 \cdots x_5)$ と表し，受信語を $D = (d_1, d_2 \cdots d_5)$ と表す．次に式 (6) について解いていく．a_1 について考えると，

$$P_{X_1|D_1}(x_1|d_1) = \sum_{x_2,x_3} P_{X|D}(x|d)$$
$$= \sum_{x_2,x_3,x_4,x_5} f(x_1, x_2)f(x_2, x_3, x_4)f(x_4, x_5)P_{\bar{X}|\bar{D}}(\bar{x}|\bar{d}) \tag{7}$$

となる．式 (7) で表されるような周辺化関数を計算するため，計算量を考えたうえで分配則により解く．そして，式 (7) をさらに解くと

$$\sum_{x_2,x_3,x_4,x_5} f_1(x_1, x_2)f_2(x_2, x_3, x_4)f_3(x_4, x_5)W(d_1|x_1)W(d_2|x_2)W(d_3|x_3)W(d_4|x_4)W(d_5|x_5) \tag{8}$$

$$= W(d_1|x_1) \times \sum_{x_2,x_3,x_4,x_5} f_1(x_1, x_2)f_2(x_2, x_3, x_4)f_3(x_4, x_5)W(d_2|x_2)W(d_3|x_3)W(d_4|x_4)W(d_5|x_5) \tag{9}$$

となる．式 (8)，(9) において，$W(d_i|x_i)$ はメッセージのやり取りを観測したシンボルを用いて，条件付き確率で表している．式 (9) では，$W(d_1|x_1)$ を和の中から出すことで，この後の計算を単純にしている．次に式 (9) の $W(d_1|x_1)$ 以外の項を展開すると

$$\sum_{x_2,x_3,x_4,x_5} f_1(x_1, x_2)f_2(x_2, x_3, x_4)f_3(x_4, x_5)W(d_2|x_2)W(d_3|x_3)W(d_4|x_4)W(d_5|x_5)$$
$$= \sum_{x_2} f_1(x_1, x_2)W(d_2|x_2) \sum_{x_3,x_4,x_5} f_2(x_2, x_3, x_4)f_3(x_4, x_5)W(d_3|x_3)W(d_4|x_4)W(d_5|x_5) \tag{10}$$

となる．式 (10) では，式 (9) の和を分解している．図 4 のファクターグラフに示されているそれぞれの関数ノードに直接送られてくるメッセージごとで分解することで、式 (7) を同様の手続きで展開する．さらに式 (10) の $\sum_{x_2} f(x_1, x_2)W(d_2|x_2)$ 以外の項を展開すると

$$\begin{aligned}
&\sum_{x_3,x_4,x_5} f_2(x_2, x_3, x_4) f_3(x_4, x_5) W(d_3|x_3) W(d_4|x_4) W(d_5|x_5) \\
&= \sum_{x_3,x_4} f_2(x_2, x_3, x_4) W(d_3|x_3) W(d_4|x_4) \times \sum_{x_5} f_3(x_4, x_5) W(d_5|x_5) \qquad (11)
\end{aligned}$$

となる．よって式 (7) を展開すると

$$\begin{aligned}
&P_{X_1|D_1}(x_1|d_1) \\
&= W(d_1|x_1) \qquad (12) \\
&\times \sum_{x_2} f(x_1, x_2) W(d_2|x_2) \qquad (13) \\
&\times \sum_{x_3,x_4} f_2(x_2, x_3, x_4) W(d_3|x_3) W(d_4|x_4) \qquad (14) \\
&\times \sum_{x_5} f_3(x_4, x_5) W(d_5|x_5) \qquad (15)
\end{aligned}$$

となる．式 (15) は，$a_5 \to f_3$ のメッセージが f_3 から a_4 に対して送られるときの結果を表している．式 (14) と式 (15) によって図 5 において，f_2 から a_2 に対して送られるメッセージを表している．同様に考え，式 (12) から式 (15) の積が，a_1 の一時推定語となる．

以上が一般的な sum-product 復号の処理の流れである．この処理が，設定されているシンボルの数で同様に行われ，すべてのシンボルで一時推定語が決定し，パリティ検査行列との積をとることで，符号語としてみなされるか判断される．次章では、今まで説明してきた無記憶通信路での一般的な通信路ではなく，記憶のある通信路について示していく．

4　本研究における sum-product 復号

この章では，本研究における sum-product 復号を用いての復号の際の問題点を述べる．そして，その問題点に対しての解決策を説明する．

4.1　記憶がある通信路

前章で説明していた sum-product 復号の計算方法は，通信路に記憶がないという仮定の下での方法であった．しかし，本研究で考えている通信路では記憶があるため復号する際に相違点がある．

まず図 8 にパケット間隔でのパケット通信について，時刻と時間間隔の関係を示す．

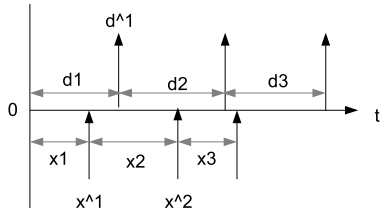

図 8　時刻と時間間隔の関係

図 8 において，横軸は時間経過を表している．図 8 において 2 種類の矢印のうち，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，時刻を示したものであり，時間軸に向かっている矢印は元の送られた符号語の時刻の様子を表している．x_1 や d_1 などはパケットの時間間隔を表しており，$\hat{x}_1$ や $\hat{d}_1$ などはパケットの到着，出発時刻を表している．この図 8 を用いて通信路に記憶がる状態と無い状態の比較を説明する．

記憶がない通信路でも，記憶のある通信路でも $\hat{d}_1$ を観測したときは．$\hat{x}_1$ のみの時刻を用いて復号すればよい．したがって必要となる時間間隔は直前の時間間隔の x_1 のみである．しかし，次に $\hat{d}_2$ の到着時刻を観測し，$\hat{x}_2$ を復号する場合を考える．まず，通信路に記憶がない場合を考える．記憶がない場合は、先ほどと同様に直前の到着間隔である x_2 のみを考える．しかし，記憶のある通信路では，以前の到着の時間間隔が影響するため x_2 だけではなく，x_1 の時間間隔も考慮する必要がある．

次に，記憶のある通信路における sum-product 復号のアルゴリズムについて，手順を示す．

具体例として，記憶のない通信路における sum-product 復号の手順の説明の時と同様に x_1 の一時推定語は

$$\begin{aligned} P_{X_1|D_1}(x_1|d_1) &= \sum_{x_2,x_{\cdot 3}} P_{X|D}(x|d) \\ &= \sum_{x_2,x_3,x_4,x_5} f(x_1,\, x_2) f(x_2, x_3,\, x_4) f(x_4,\, x_5) P_{\bar{X}|\bar{D}}(\bar{x}|\bar{d}) \\ &= \sum_{x_2,x_3,x_4,x_5} f_1(x_1,\, x_2) f_2(x_2, x_3,\, x_4) f_3(x_4,\, x_5) W(\hat{d}_1|\hat{x}_1) W(\hat{d}_2|\hat{x}_2) W(\hat{d}_3|\hat{x}_3) W(\hat{d}_4|\hat{x}_4) W(\hat{d}_5|\hat{x}_5) \end{aligned} \tag{16}$$

となる．

式 (16) までの手順は，記憶のない通信路と変わらないように見える．しかし，メッセージの内容が時間間隔ではなく時刻で書かれている．記憶のない通信路では，考えている到着の以前

の到着を考える必要がないと説明した．よって計算の手順の中でも時間間隔で計算することができた．しかし，本研究で扱っている記憶のある通信路では，以前の到着も考える必要があるため，式 (8) と同様の形で書くには時刻で考える必要がある．以下に式 (16) を展開すると

$$\begin{aligned}&\sum_{x_2,x_3,x_4,x_5} f_1(x_1,\ x_2)f_2(x_2,x_3,\ x_4)f_3(x_4,\ x_5)W(\hat{d}_1|\hat{x}_1)W(\hat{d}_2|\hat{x}_2,\hat{d}_1)W(\hat{d}_3|\hat{x}_3,\hat{d}_2)W(\hat{d}_4|\hat{x}_4,\hat{d}_3)W(\hat{d}_5|\hat{x}_5,\hat{d}_4)\\&= W(d_1|x_1)\times\sum_{x_2,,x_3,x_4,x_5} f_1(x_1,\ x_2)f_2(x_2,x_3,\ x_4)f_3(x_4,\ x_5)W(\hat{d}_2|\hat{x}_2,\hat{d}_1)W(\hat{d}_3|\hat{x}_3,\hat{d}_2)W(\hat{d}_4|\hat{x}_4,\hat{d}_3)W(\hat{d}_5|\hat{x}_5,\hat{d}_4)\end{aligned} \tag{17}$$

となる．式 (17) では，メッセージが条件付確率で書かれているが条件の内容が記憶のない通信路とは異なっている．記憶のない通信路と同様に式 (17) を展開すると

$$\begin{aligned}&\sum_{x_2,,x_3,x_4,x_5} f_1(x_1,\ x_2)f_2(x_2,x_3,\ x_4)f_3(x_4,\ x_5)W(\hat{d}_2|\hat{x}_2,\hat{d}_1)W(\hat{d}_3|\hat{x}_3,\hat{d}_2)W(\hat{d}_4|\hat{x}_4,\hat{d}_3)W(\hat{d}_5|\hat{x}_5,\hat{d}_4)\\&=\sum_{x_2} f_1(x_1,x_2)W(\hat{d}_2|\hat{x}_2,\hat{d}_1)\sum_{x_3,x_4,x_5} f_2(x_2,x_3,\ x_4)f_3(x_4,\ x_5)W(\hat{d}_3|\hat{x}_3,\hat{d}_2)W(\hat{d}_4|\hat{x}_4,\hat{d}_3)W(\hat{d}_5|\hat{x}_5,\hat{d}_4)\end{aligned} \tag{18}$$

となる．式 (18) の内側の和を先ほどと同様に展開すると

$$\begin{aligned}&\sum_{x_3,x_4,x_5} f_2(x_2,x_3,\ x_4)f_3(x_4,\ x_5)W(\hat{d}_3|\hat{x}_3,\hat{d}_2)W(\hat{d}_4|\hat{x}_4,\hat{d}_3)W(\hat{d}_5|\hat{x}_5,\hat{d}_4)\\&=\sum_{x_3,x_4} f_2(x_2,x_3,\ x_4)W(\hat{d}_3|\hat{x}_3,\hat{d}_2)W(\hat{d}_4|\hat{x}_4,\hat{d}_3)\times\sum_{x_5} f_3(x_4,\ x_5)W(\hat{d}_5|\hat{x}_5,\hat{d}_4) \qquad (19)\\&=\sum_{x_3,x_4} f_2(x_2,x_3,\ x_4)W(\hat{d}_3|x_1+x_2+x_3,\hat{d}_2)W(\hat{d}_4|x_1+x_2+x_3+x_4,\hat{d}_3)\\&\times\sum_{x_5} f_3(x_4,\ x_5)W(\hat{d}_5|x_1+x_2+x_3+x_4+x_5,\hat{d}_4) \qquad (20)\end{aligned}$$

となる．式 (19) は，式 (16) を展開した一部を表している．式 (19) では，メッセージの中身を時刻で表している．しかし，記憶のある通信路では以前の到着時間も影響するため，時刻表記しているメッセージを時間間隔に書き換えて考える必要がある．そして，時間間隔で示した式を式 (20) で示している．ここで本研究における問題点が発生している．式 (20) の和は x_3，x_4 や x_5 で回している．しかし，どちらの和においてもメッセージの中に x_3，x_4，x_5 以外の x_1，x_2 などが存在している．よって，この計算を行った際には，これらの関数が関数のまま計算結果に残ってしまい，計算を完了することが出来ない．そして，3 章 2 節でツリー構造と sum-product 復号の計算の対応を説明したが，記憶のある通信路では sum-product 復号の計算が完了しないため，対応するツリー構造を書くことができない．これは x_1 のみではなく，他のシンボルでも同様なことが言える．よって，本研究で扱うような記憶のある通信路ではツリー構造を書き，sum-product 復号の計算の流れを確認する通常の考え方ができない．

そこで，次節でこの問題点に対して解決策を提示する．

4.2 解決策

記憶のある通信路で sum-product 復号をツリー構造を用いて行うための策は，それぞれのシンボルに初期値を設けることである．この初期値は図 9 を用いて決定することができる．

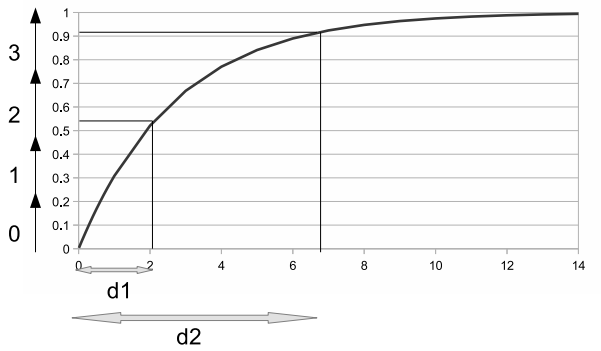

図 9　仮のシンボルと時間間隔の関係

図 9 は，指数分布の分布関数を表している．そして，x 軸は時間間隔を表しており，y 軸は一様分布で割り当てられる初期値を表している．パケット間隔を用いたパケット通信は，指数分布で符号を発生させると最も効率が良いとされているため，指数分布を用いている．受信語として得られた時間間隔に対する累積確率を求め, 対応するシンボルを初期値とする．仮に，時間間隔が 6 なら，x 軸の 6 から分布関数によって対応するシンボルが初期値になるため 3 という初期値が割り当てられる．同様に時間間隔が 2 なら，初期値として 2 が割り当てられる．このような操作を最初のシンボルには行う. 時間間隔と初期値の関係式は

$$\lfloor (1 - e^{-\lambda d}) \times 4 \rfloor = x \tag{21}$$

となる. 式 (21) の d に受信した時間間隔を代入することで，初期値を求めることができる．式 (21) において，λ は指数分布のパラメータ，d は確率変数である．

この操作によって，1 回目の反復処理が行われ 1 回目の各シンボルの一時推定語が導き出される．もしこの一時推定語が，符号語として見なされなかった場合は，この一時推定語を 2 回目の反復復号の際の各シンボルの初期値として扱う．よって，図 9 を用いて初期値を求めるのは反復復号を行う最初のみに限られている．本研究で提案した方法では，厳密には実際に考えるべき計算はせずに，近似した計算を行っている．実際には，式 (16) を計算することも可能だが，大変複雑であり手間がかかる．また，sum-product 復号は可能だが，前提のようにツリー構造を書いたうえで，sum-product 復号を行うことができない．実際に例を考えると，式 (20) の中に近似的に求められたシンボルを代入すると

$$\sum_{x_3,x_4} f_2(x_2,x_3,\,x_4)W(\hat{d}_3|s_3,\hat{d}_2)W(\hat{d}_4|s_4,\hat{d}_3) \times \sum_{x_5} f_3(x_4,\,x_5)W(\hat{d}_5|s_5,\hat{d}_4) \tag{22}$$

のようになる．式 (22) の s_3，s_4，s_5 はそれぞれ提案した方法によって求められた近似的なシンボルであり，定数なので計算することができる．この計算結果が，元々の本当の計算式を計算した結果と同様の値をとるかは考察することができていないため，今後考える必要がある．

5　結果と考察

この章では，提案した復号法で符号語が出力されるのかとプログラムの中で工夫した点を述べる．そして，最後に今後の展望を述べる．

5.1　結果

今までに説明してきたアルゴリズムでプログラムを作った．プログラムは, アイデア通りに作った最初のプログラムと繰り返し回数の削減を狙った改良版プログラムの 2 つがある. 改良版の内容については後で述べる．　具体的なプログラムは付録に記載している．そして，プログラムを実行した結果は図 12 に表として記載する．本研究でのプログラムでは，サービスレートは一般性を損なうことなく $\mu = 1$ とする．いくつかの受信語に対し，そのパケット間隔で復号した際に，式 (2) で示したパリティ検査行列と推定語の積をとることで，符号語として見なされるかどうかについての結果を図 12 に示す．符号語として見なされるには，積の結果が 0 になれば良い．本研究では符号の良し悪しは考えていないがパリティ検査行列を変えることで符号の性能をよくすることができる．3 列目と 4 列目の復号シンボルについては，1 回目の復号で推定された復号シンボルと sum-product 復号の結果が収束した際の結果について表している．5 列目の試行回数は，最初の数字が最初のプログラムを用いた際に，結果が収束するまでに復号処理を行った回数であり，右側の数字が改良版プログラムを用いた結果である．図 12 に掲載している結果は一部の結果であるが，改良版プログラムを用いることで試行回数が減少していることが分かる．本研究では，シンボルの数が少ないが，今後増やしていった際には，最初のプログラムに比べて，より少ない回数で復号を行うことができる．そして，符号語かどうかの判断については，図 12 で表されたパケット間隔の半分ほどが符号語ではない判断されている．本研究では，符号語の構成については，考察することはできていないため，記憶のある通信路で sum-product 復号を行う際の符号語の構成については，今後の課題である．

前章で説明した初期値を設ける方法によって，sum-product 復号を行うことができた．本研

究では，提案したアルゴリズムで作成したプログラムによって符号語が生成されれば復号ができたと判定している．よって，この復号方法などの誤り確率などの精度に関しては考察出来ていない．そして，このプログラムの中でも工夫することによって，符号語が発生されるまで，あるいは結果が収束するまでのメッセージの交換回数を短縮することができた．図 10，図 11 にプログラムの中の工夫点についてを表している．本研究で考えている処理手順では通常は図 10 のように考える．図 10 では 2 回目の更新において必要となる要素である $x_1 \cdots x_5$ は 1 回目の $x_1 \cdots x_5$ を用いて，2 回目の反復復号を行っていた．しかし，本研究では 2 回目の反復復号の際も 1 回目の一時推定語を用いるのは x_1 のみである．その他は，2 回目の反復復号で一時推定語となったシンボルをそのまま用いる．図 11 にその様子を描いている．前章で説明したように，本研究では一時推定語を求める際には，以前の到着間隔も影響する．よって，x_2 の一時推定語を求めるには x_1 が影響し，x_3 をの一時推定語を求める際には x_1，x_2 が影響する．これを踏まえて，本研究では 2 回目の x_2 一時推定語を求める際に，今までは 1 回目の x_1 を使用していたのを 2 回目の x_1 の一時推定語を用いることにした．同様に，2 回目の x_3 の一時推定語を求める際には，1 回目の x_1，x_2 を用いるので無く，2 回目の x_1，x_2 を用いた．この結果，一時推定語の結果が収束するまでの試行回数を少なくすることが出来た．そして，これに伴い sum-product 復号にかかる時間も，通常の考えよりは短くすることができた．

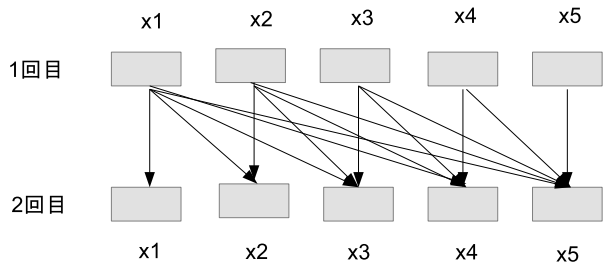

図 10　最初のアルゴリズム

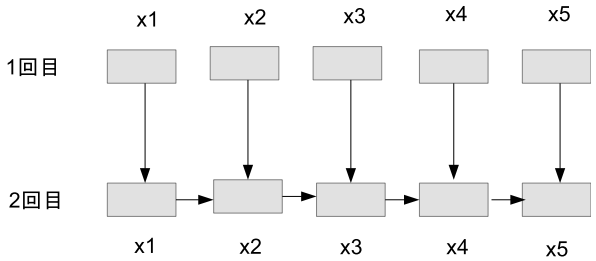

図 11　改良版アルゴリズム

パケット間隔	符号語の判断	最初の復号シンボル	最終的な復号シンボル	試行回数
[0, 0.5, 1, 1.5, 2, 2.5]	符号語である	{0, 0, 0, 0, 0}	{0, 0, 0, 0, 0}	1回→1回
[0, 1, 2, 3, 4, 5]	符号語である	{1, 1, 1, 1, 1}	{1, 1, 1, 1, 1}	2回→1回
[0, 2.5, 5, 7.5, 10, 12.5]	符号語ではない	{2, 2, 2, 2, 2}	{1, 1, 2, 2, 2}	6回→4回
[0, 3.5, 7, 10.5, 14, 17.5]	符号語ではない	{2, 2, 2, 2, 2}	{2, 2, 2, 3, 2}	3回→2回
[0, 4, 7, 9, 10, 10.5]	符号語ではない	{3, 2, 2, 1, 0}	{2, 2, 2, 1, 0}	6回→4回
[0, 0.1, 0.5, 1.0, 2.5,6.0]	符号語である	{0, 0, 0, 1, 2}	{0, 0, 0, 0, 0}	3回→2回
[0, 1.1, 1.4, 2.2, 4.0, 6.2]	符号語である	{1, 0, 1, 1, 2}	{0, 0, 1, 1, 1}	4回→3回
[0, 2.4, 4.1, 5.2, 8.4, 11.2]	符号語ではない	[2, 1, 1, 2, 2}	{1, 1, 0, 3, 3}	5回→3回
[0, 2.2, 5.5, 6.1, 7.4, 9.0]	符号語である	[2, 2, 0, 1, 1}	{1, 1, 0, 1, 1}	2回→1回

図 12　本研究のプログラムの結果

5.2　今後の展望

本研究で提案した解決策により記憶のある通信路で sum-product 復号を行うことができた．しかし，まだ可能になったというだけで性能については考察することができていない．そこで，この考え方が実際に効率が良いのか，そしてもっと性能の良い考え方はないのかなど考えていく必要がある．そして，通信路の記憶の有無がどれほど LDPC 符号と sum-product 復号の組み合わせに影響するかを考察出来れば，符号語数，シンボル数などの影響度も調べていきたい．

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”, Transactions on Information Theory, Vol. 42,pp.4–18, 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] 飯島一貴,「パケット間隔で情報を送る通信路に対する誤り訂正符号の構成に関する考察」, 信州大学大学院総合理工学研究科修士論文 (指導教員：西新幹彦), 2017 年 3 月.

[4] 和田山正， 誤り訂正技術の基礎， 森北出版， 2010.

付録 A　ソースコード

A.1　本研究で提案した案のプログラム

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


double expdistfunc(double x, double rate)
{
    return(1 - exp(-x * rate));

}

double expdistinv(double x, double rate)
{
    return(-log(1 - x) / rate);

}

double unidistfunc(int x)
{
    return((x + 0.5) / 4);
}

double channel(double dep, double arr, double pdep)
{
    double servicetime;

    if (dep <= arr) {
        return(0.0);
    }
    servicetime = dep - max(arr, pdep);
    return(exp(-servicetime));
}


int main()
{
    double dep[6] = { 0, 0.3, 1.3, 2.7, 8.6, 28.0 };
    int res[5];
    double arr[6];
    double narr[6];
    double exp = 2.7182818;
    double symbolnum[6];
    int i;
    int p;
    int q;
    int b;
    int s;
    int a[3][5];
    int det = 0;
    double checkmat[3][5] = { {1, 1, 0, 0, 0}, {0, 1, 1, 1, 0}, {0, 0, 0, 1, 1} };
    double dsymbol[5][4];
    double dmessage[5][4];
    double message[5][4];
    double messageaf[7][4];
    double narrtime[6];
    double arrtime[6];
    double messagefa[7][4];
    double bra[6];
    double brabox[7][2];
    int start[7] = { 1, 2, 2, 3, 4, 4, 5 };
```

```
int finish[7] = { 1, 1, 2, 2, 2, 3, 3 };
double max;
double max1;
double change[4];
int maxb;
int maxbb[5];
int bbq[3];
int x;
int nn;


for (i = 0; i < 5; i++) {
    //時間間隔からシンボル//
    res[i] = expdistfunc(dep[i + 1] - dep[i], 1 / exp) * 4;
}

printf("復号シンボル : %d, %d, %d, %d, %d\n", res[0], res[1], res[2], res[3], res[4]);

for (i = 0; i < 5; i++) {
    //シンボルから間隔//
    symbolnum[i] = unidistfunc(res[i]);
}
printf("シンボルの値 : %f, %f, %f, %f, %f\n", symbolnum[0], symbolnum[1], symbolnum[2], symbolnum[3], symbolnum[4]);

for (i = 0; i < 5; i++) {
    arr[i] = expdistinv(symbolnum[i], 1 / exp);

}
printf("時間間隔 : %f, %f, %f, %f, %f\n", arr[0], arr[1], arr[2], arr[3], arr[4]);

arrtime[0] = 0;
arrtime[1] = arr[0];
arrtime[2] = arr[0] + arr[1];
arrtime[3] = arr[0] + arr[1] + arr[2];
arrtime[4] = arr[0] + arr[1] + arr[2] + arr[3];
arrtime[5] = arr[0] + arr[1] + arr[2] + arr[3] + arr[4];
printf("時刻 : %f, %f, %f, %f, %f, %f\n", arrtime[0], arrtime[1], arrtime[2], arrtime[3], arrtime[4], arrtime[5]);

for (p = 0; p < 7; p++) {
    for (q = 0; q < 4; q++) {
        message[p][q] = channel(dep[start[p]], (-log(1 - (2 * q + 1) / 8.0)) / (1 / exp), dep[start[p] - 1]);

    }
}


printf("初期値 : %f, %f, %f\n", message[0][1], message[1][0], message[1][1]);

messagefa[0][0] = message[1][0];
messagefa[0][1] = message[1][1];
messagefa[0][2] = message[1][2];
messagefa[0][3] = message[1][3];
messagefa[1][0] = message[0][0];
messagefa[1][1] = message[0][1];
messagefa[1][2] = message[0][2];
messagefa[1][3] = message[0][3];
messagefa[5][0] = message[6][0];
messagefa[5][1] = message[6][1];
messagefa[5][2] = message[6][2];
messagefa[5][3] = message[6][3];
messagefa[6][0] = message[5][0];
messagefa[6][1] = message[5][1];
messagefa[6][2] = message[5][2];
messagefa[6][3] = message[5][3];

printf("messagefa : %f\n", messagefa[1][2]);
printf("message :  %f, %f, %f, %f, %f, %f, %f, %f\n", message[2][0], message[4][0], message[2][1], message[4][1], mess

messagefa[2][0] = message[3][0] * message[4][0] + message[3][1] * message[4][1] + message[3][2] * message[4][2] + mess
```

```
messagefa[2][1] = message[3][0] * message[4][1] + message[3][1] * message[4][0] + message[3][2] * message[4][3] + mess
messagefa[2][2] = message[3][0] * message[4][2] + message[3][1] * message[4][3] + message[3][2] * message[4][0] + mess
messagefa[2][3] = message[3][0] * message[4][3] + message[3][1] * message[4][2] + message[3][2] * message[4][1] + mess

messagefa[3][0] = message[2][0] * message[4][0] + message[2][1] * message[4][1] + message[2][2] * message[4][2] + mess
messagefa[3][1] = message[2][0] * message[4][1] + message[2][1] * message[4][0] + message[2][2] * message[4][3] + mess
messagefa[3][2] = message[2][0] * message[4][2] + message[2][1] * message[4][3] + message[2][2] * message[4][0] + mess
messagefa[3][3] = message[2][0] * message[4][3] + message[2][1] * message[4][2] + message[2][2] * message[4][1] + mess

messagefa[4][0] = message[3][0] * message[2][0] + message[3][1] * message[2][1] + message[3][2] * message[2][2] + mess
messagefa[4][1] = message[3][0] * message[2][1] + message[3][1] * message[2][0] + message[3][2] * message[2][3] + mess
messagefa[4][2] = message[3][0] * message[2][2] + message[3][1] * message[2][3] + message[3][2] * message[2][0] + mess
messagefa[4][3] = message[3][0] * message[2][3] + message[3][1] * message[2][2] + message[3][2] * message[2][1] + mess

printf("messagefa : %f\n", messagefa[2][2]);
//a1 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[0][b] = channel(dep[1], (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[0]) * messagefa[0][b];
}

printf("dsymbol1 : %f, %f, %f, %f\n", dsymbol[0][0], dsymbol[0][1], dsymbol[0][2], dsymbol[0][3]);



max1 = dsymbol[0][0];
maxb = 0;
for (b = 0; b < 4; b++) {
    if (max1 < dsymbol[0][b]) {
        max1 = dsymbol[0][b];
        maxb = b;
    }
}


maxbb[0] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[0]);

//a2 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[1][b] = channel(dep[2], arrtime[1] + (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[1]) * messagefa[1][b]
}

printf("dsymbol2 : %f, %f, %f, %f\n", dsymbol[1][0], dsymbol[1][1], dsymbol[1][2], dsymbol[1][3]);

max1 = dsymbol[1][0];
maxb = 0;
for (b = 1; b < 4; b++) {
    if (max1 < dsymbol[1][b]) {
        max1 = dsymbol[1][b];
        maxb = b;
    }
}

maxbb[1] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[1]);




//a3 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[2][b] = channel(dep[3], arrtime[2] + (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[2]) * messagefa[3][b];
}
```

```
printf("dsymbol3 : %f, %f, %f, %f\n", dsymbol[2][0], dsymbol[2][1], dsymbol[2][2], dsymbol[2][3]);

max1 = dsymbol[2][0];
maxb = 0;
for (b = 1; b < 4; b++) {
    if (max1 < dsymbol[2][b]) {
        max1 = dsymbol[2][b];
        maxb = b;
    }
}

maxbb[2] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[2]);



//a4 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[3][b] = channel(dep[4], arrtime[3] + (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[3]) * messagefa[4][b]
}

printf("dsymbol4 : %g, %g, %g, %g\n", dsymbol[3][0], dsymbol[3][1], dsymbol[3][2], dsymbol[3][3]);

max1 = dsymbol[3][0];
maxb = 0;
for (b = 1; b < 4; b++) {
    if (max1 < dsymbol[3][b]) {
        max1 = dsymbol[3][b];
        maxb = b;
    }

}

maxbb[3] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[3]);



//a5 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[4][b] = channel(dep[5], arrtime[4] + (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[4]) * messagefa[6][b];
}

printf("dsymbol5 : %g, %g, %g, %g\n", dsymbol[4][0], dsymbol[4][1], dsymbol[4][2], dsymbol[4][3]);

max1 = dsymbol[4][0];
maxb = 0;
for (b = 1; b < 4; b++) {
    if (max1 < dsymbol[4][b]) {
        max1 = dsymbol[4][b];
        maxb = b;
    }
}

maxbb[4] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[4]);
```

```
bbq[0] = maxbb[0] ^ maxbb[1];
bbq[1] = maxbb[1] ^ maxbb[2] ^ maxbb[3];
bbq[2] = maxbb[3] ^ maxbb[4];

printf("符号語のチェック = %d, %d, %d\n", bbq[0], bbq[1], bbq[2]);

if (bbq[0] == 0 && bbq[1] == 0 && bbq[2] == 0) {
    printf("符号語とみなす!\n", bbq);
}
else {
    printf("符号語ではない！\n", bbq);
}


//2 回目の更新
for (x = 0; x < 10; x++) {
    for (i = 0; i < 5; i++) {
        //シンボルから間隔//
        symbolnum[i] = unidistfunc(maxbb[i]);
    }
    printf("シンボルの値 : %f, %f, %f, %f, %f\n", symbolnum[0], symbolnum[1], symbolnum[2], symbolnum[3], symbolnum[4]);

    for (i = 0; i < 5; i++) {
        arr[i] = expdistinv(symbolnum[i], 1 / exp);

    }
    printf("時間間隔 : %f, %f, %f, %f, %f\n", arr[0], arr[1], arr[2], arr[3], arr[4]);

    arrtime[0] = 0;
    arrtime[1] = arr[0];
    arrtime[2] = arr[0] + arr[1];
    arrtime[3] = arr[0] + arr[1] + arr[2];
    arrtime[4] = arr[0] + arr[1] + arr[2] + arr[3];
    arrtime[5] = arr[0] + arr[1] + arr[2] + arr[3] + arr[4];
    printf("時刻 : %f, %f, %f, %f, %f, %f\n", arrtime[0], arrtime[1], arrtime[2], arrtime[3], arrtime[4], arrtime[5]);


    //[0][3][6] の message
    for (q = 0; q < 4; q++) {
        message[0][q] = channel(dep[start[0]], (-log(1 - (2 * q + 1) / 8.0)) / (1 / exp), dep[start[0] - 1]);
        message[3][q] = channel(dep[start[3]], (-log(1 - (2 * q + 1) / 8.0)) / (1 / exp), dep[start[3] - 1]);
        message[6][q] = channel(dep[start[6]], (-log(1 - (2 * q + 1) / 8.0)) / (1 / exp), dep[start[6] - 1]);
    }

    //[1][2][4][5] の message
    for (q = 0; q < 4; q++) {
        message[1][q] = channel(dep[start[1]], (-log(1 - (2 * q + 1) / 8.0)) / (1 / exp), dep[start[1] - 1]) * message
        message[2][q] = channel(dep[start[2]], (-log(1 - (2 * q + 1) / 8.0)) / (1 / exp), dep[start[2] - 1]) * message
        message[4][q] = channel(dep[start[4]], (-log(1 - (2 * q + 1) / 8.0)) / (1 / exp), dep[start[4] - 1]) * message
        message[5][q] = channel(dep[start[5]], (-log(1 - (2 * q + 1) / 8.0)) / (1 / exp), dep[start[5] - 1]) * message
    }

    printf("新しいメッセージ : %f, %f, %f\n", message[0][1], message[1][0], message[1][1]);


    messagefa[0][0] = message[1][0];
    messagefa[0][1] = message[1][1];
    messagefa[0][2] = message[1][2];
    messagefa[0][3] = message[1][3];
    messagefa[1][0] = message[0][0];
    messagefa[1][1] = message[0][1];
    messagefa[1][2] = message[0][2];
    messagefa[1][3] = message[0][3];
    messagefa[5][0] = message[6][0];
    messagefa[5][1] = message[6][1];
    messagefa[5][2] = message[6][2];
    messagefa[5][3] = message[6][3];
    messagefa[6][0] = message[5][0];
    messagefa[6][1] = message[5][1];
```

```
messagefa[6][2] = message[5][2];
messagefa[6][3] = message[5][3];

printf("messagefa : %f\n", messagefa[1][1]);
printf("message :  %f, %f, %f, %f, %f, %f, %f, %f\n", message[1][0], message[3][0], message[1][1], message[3][1],

messagefa[2][0] = message[3][0] * message[4][0] + message[3][1] * message[4][1] + message[3][2] * message[4][2] +
messagefa[2][1] = message[3][0] * message[4][1] + message[3][1] * message[4][0] + message[3][2] * message[4][3] +
messagefa[2][2] = message[3][0] * message[4][2] + message[3][1] * message[4][3] + message[3][2] * message[4][0] +
messagefa[2][3] = message[3][0] * message[4][3] + message[3][1] * message[4][2] + message[3][2] * message[4][1] +

messagefa[3][0] = message[2][0] * message[4][0] + message[2][1] * message[4][1] + message[2][2] * message[4][2] +
messagefa[3][1] = message[2][0] * message[4][1] + message[2][1] * message[4][0] + message[2][2] * message[4][3] +
messagefa[3][2] = message[2][0] * message[4][2] + message[2][1] * message[4][3] + message[2][2] * message[4][0] +
messagefa[3][3] = message[2][0] * message[4][3] + message[2][1] * message[4][2] + message[2][2] * message[4][1] +

messagefa[4][0] = message[3][0] * message[2][0] + message[3][1] * message[2][1] + message[3][2] * message[2][2] +
messagefa[4][1] = message[3][0] * message[2][1] + message[3][1] * message[2][0] + message[3][2] * message[2][3] +
messagefa[4][2] = message[3][0] * message[2][2] + message[3][1] * message[2][3] + message[3][2] * message[2][0] +
messagefa[4][3] = message[3][0] * message[2][3] + message[3][1] * message[2][2] + message[3][2] * message[2][1] +

//a1 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[0][b] = channel(dep[1], (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[0]) * messagefa[0][b];
}

printf("dsymbol1 : %f, %f, %f, %f\n", dsymbol[0][0], dsymbol[0][1], dsymbol[0][2], dsymbol[0][3]);



max1 = dsymbol[0][0];
maxb = 0;
for (b = 0; b < 4; b++) {
    if (max1 < dsymbol[0][b]) {
        max1 = dsymbol[0][b];
        maxb = b;
    }
}

maxbb[0] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[0]);



//a2 の尤度

for (b = 0; b < 4; b++) {
    dsymbol[1][b] = channel(dep[2], arrtime[1] + (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[1]) * messagefa[1]
}

printf("dsymbol2 : %f, %f, %f, %f\n", dsymbol[1][0], dsymbol[1][1], dsymbol[1][2], dsymbol[1][3]);

max1 = dsymbol[1][0];
maxb = 0;
for (b = 1; b < 4; b++) {
    if (max1 < dsymbol[1][b]) {
        max1 = dsymbol[1][b];
        maxb = b;
    }
}

maxbb[1] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[1]);
```

```
//a3 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[2][b] = channel(dep[3], arrtime[2] + (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[2]) * messagefa[3]
}

printf("dsymbol3 : %f, %f, %f, %f\n", dsymbol[2][0], dsymbol[2][1], dsymbol[2][2], dsymbol[2][3]);

max1 = dsymbol[2][0];
maxb = 0;
for (b = 1; b < 4; b++) {
    if (max1 < dsymbol[2][b]) {
        max1 = dsymbol[2][b];
        maxb = b;
    }
}

maxbb[2] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[2]);



//a4 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[3][b] = channel(dep[4], arrtime[3] + (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[3]) * messagefa[4]
}

printf("dsymbol4 : %g, %g, %g, %g\n", dsymbol[3][0], dsymbol[3][1], dsymbol[3][2], dsymbol[3][3]);

max1 = dsymbol[3][0];
maxb = 0;
for (b = 1; b < 4; b++) {
    if (max1 < dsymbol[3][b]) {
        max1 = dsymbol[3][b];
        maxb = b;
    }
}

maxbb[3] = maxb;

printf("尤度が最も高い = %g\n", max1);
printf("シンボル = %d\n", maxb);
printf("シンボル = %d\n", maxbb[3]);




//a5 の尤度
for (b = 0; b < 4; b++) {
    dsymbol[4][b] = channel(dep[5], arrtime[4] + (-log(1 - (2 * b + 1) / 8.0)) / (1 / exp), dep[4]) * messagefa[6]
}

printf("dsymbol5 : %g, %g, %g, %g\n", dsymbol[4][0], dsymbol[4][1], dsymbol[4][2], dsymbol[4][3]);

max1 = dsymbol[4][0];
maxb = 0;
for (b = 1; b < 4; b++) {
    if (max1 < dsymbol[4][b]) {
        max1 = dsymbol[4][b];
        maxb = b;
    }
}

maxbb[4] = maxb;
```

```
        printf("尤度が最も高い = %g\n", max1);
        printf("シンボル = %d\n", maxb);
        printf("シンボル = %d\n", maxbb[4]);


        bbq[0] = maxbb[0] ^ maxbb[1];
        bbq[1] = maxbb[1] ^ maxbb[2] ^ maxbb[3];
        bbq[2] = maxbb[3] ^ maxbb[4];

        printf("符号語のチェック = %d, %d, %d\n", bbq[0], bbq[1], bbq[2]);

        if (bbq[0] == 0 && bbq[1] == 0 && bbq[2] == 0) {
            printf("符号語とみなす!\n", bbq);
        }
        else {
            printf("符号語ではない！\n", bbq);
        }

        x;
    }
    getchar();
    return(0);
}
```