信州大学工学部

学士論文

ポアソン到着シンボルに対する
タンストール符号の遅延発生メカニズムの解析

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　06T2801F
氏名　泉　陽一

2010 年 3 月 19 日

目次

1　序章

1.1　背景

インターネットや携帯電話などの情報伝送システムにおいてそのデータ量は莫大である．その莫大なデータを誤りがなくかつ効率よく伝送するにはどうするべきかが問題となる．本研究ではデータを圧縮し, かつ遅延を最小限にしてデータを送ることを考える．符号化方法にはいくつかあるが, 定常無記憶情報源に対して最適な VF 符号 (variable-length-to-fixed-length code) であるタンストール符号 (Tunstall code) と一般的に良く使われる最適な FV 符号 (fixed-length-to-variable-lengthcode) であるハフマン符号とで遅延に大きな差がないのであれば VF 符号はノイズの影響を受けにくいという特徴があるので, タンストール符号はハフマン符号よりもデータを誤りがなくかつ効率よく伝送するのに適しているといえる．そこで, 本研究では 2 つの符号語で遅延を比較しタンストール符号はハフマン符号より優れた符号語であるのかを調べることを本研究の目的とする．まずはじめに本研究の概要について述べる．

1.2　研究の概要

本研究では図 1 に示すタンストール符号の伝送システムを用いてシステムの遅延を求めハフマン符号のシステムと遅延を比較する．まず, 情報源は 0,1 の 2 種類のシンボルを到着レート

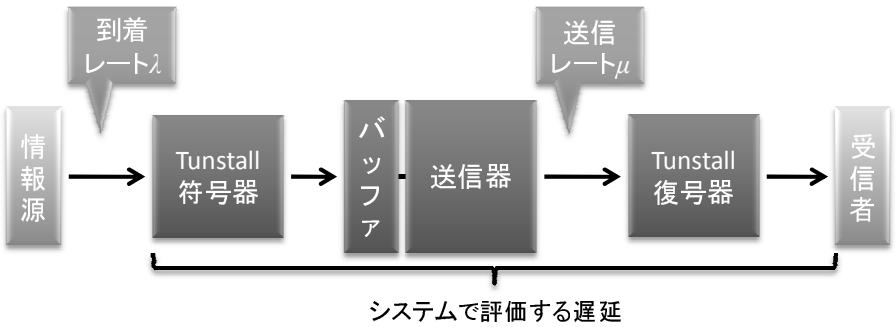

図 1　Tunstall 符号のシステム

λ [シンボル/秒] のポアソン到着で出力する．タンストール符号器に入力されたシンボルは符号語が完成するまで符号器で待たされることになる．完成した符号語は送信機に送られるが, 前の符号語が送信中の場合はバッファで待たされる．そして, 送信機から送信レート μ [ビッ

ト/秒] で復号器に送られる．このシステムで考える遅延は符号器の待ち時間とバッファでの待ち時間, 送信時間の 3 種類ある．この 3 つの遅延を合わせたものがシステム全体の遅延である．到着レート λ と送信レート μ の両方を大きくしても伝送システム全体の処理速度を速くするだけなので, 一般性を損なうことなく送信レート μ は 1 [ビット/秒] に固定する．従来研究に算術符号を用いたシステムとハフマン符号を用いたシステムがあるが, 本研究は従来研究の符号器と復号器の部分をタンストール符号器, 復号器に置き換えたものである. 算術符号は 1 シンボルずつ符号化していくシステムであり, ハフマン符号はシンボルをブロック化して符号化するものであるが, タンストール符号はシンボルを可変長の長さで区切り符号化を行う．

2 遅延

伝送システムを用いて情報を送る場合, 遅延は必ず発生する. 遅延はできるだけ小さいほうが情報を送る側にとっても受け取るほうにとっても望ましいと言える．つまり, 遅延の少ないシステムのほうがより優れたシステムであると言える. 本研究における遅延とは, ある 1 つのシンボルがエンコーダーに入力されて符号語となり送信バッファに送られて, そこからデコーダに送られシンボルが復元されるまでに要する時間のことである．

3 ポアソン到着

ポアソン到着とはランダムに到着するシンボルを表す確率過程のひとつである. ポアソン到着のシンボルの到着間隔は指数分布にしたがう．ポアソン到着はシンボルの到着が次の 3 つの条件を満たすと仮定した 1 つの到着の仕方のモデルである．

- 定常性
  $t > 0$,$x \geq 0$ なる任意の実数 t, 任意の整数 x に対し, 時間間隔 $(a, a+t)$ の間に x 人到着する確率は, 全ての a について同一で, $v_x(t)$ と表すことができる. つまり, 時間区間の幅が同じであれば, どこをとってもシステムの確率的状態は同じである. この性質を定常性という．
- 独立性
  上記に記した $v_x(t)$ は, 時刻 a までにどれだけきたか, またいつ来たかには無関係である. もし前に到着したシンボルの時刻や個数によってこの確率 $v_x(t)$ が影響を受けるならば, 前のシンボルはあとに残る影響, つまり残留効果を持つことになるが, そういうことがないというのがこの独立性である．
- 希少性

$\psi(t)$=$\sum_{x=2}^{\infty} v_x(t)$ と置くとき,t が十分小さければ $\psi(t)$ は無視してよい. つまり,

$$\psi(t) = \mathrm{o}\ (t) \quad (t \to 0) \tag{1}$$

が成り立つ. この式は,

$$\lim_{t \to 0} \frac{\psi(t)}{t} = 0 \tag{2}$$

を意味する.$o(t)$ とは t が十分小さければ $\psi(t)$ は無視してよい程度であることを示している. つまり $\psi(t) = o(t)$ は,2 つ以上のシンボルが同時に連れだってこないことを意味する.

4 タンストール符号について

4.1 VF 符号

VF 符号 (variable-length-to-fixed-length code) とは名前の通り入力の情報源ブロックの長さが可変長であり, 出力の符号語長が一定長である符号語である. 符号語長が一定であるため符号語に誤りが発生したとしても復号したシンボル列に誤りが伝搬することがないといった実用上好ましい性質がある.

4.2 タンストール符号の符号化

定常無記憶情報源に対して最適な VF 符号は発案者の名にちなんで Tunstall 符号と呼ばれる. タンストール符号の構成もハフマン符号と同じく木 (tree) を使う. ハフマン符号との違いは符号語アルファベットの代わりに情報源アルファベットに含まれるシンボルで木にラベル付けを行う. 構成法は次のとおりである.

1. 高さが 1 で情報源アルファベットの大きさと同じ数の枝をもつ木を T_1^* とおく.
2. 各 $i = 2, ..., m$ に対して, 以下を繰り返す.
（a）木の葉の中から, 最大の確率をもつ葉を新芽 (sprout) とする.
（b）（a）で決めた新芽に T_1^* を接ぎ木する.
3. 木が完成したら, 各葉に符号語をラベル付けする.

上記の構成法で作られた内部ノードの数が m 個の木 (タンストール木) を T_m^* とする. 図 2 にてタンストール木の構成の例を示す.

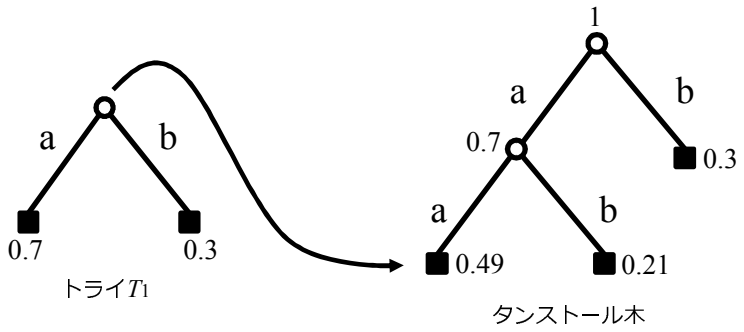

図 2　タンストール木の構成法

4.3　タンストール符号の復号

復号器も符号器と同じ木を使って, 受け取った符号語と同じ符号語の与えられた葉からルートに向かって進む過程でとおった枝のラベルをルートに近い順に並べ替えて復号する.

4.4　タンストール符号の平均符号化レート

内部ノード数が m のタンストール木 T の平均符号化レートを r_m と定義すると, 平均符号化レート $r_m(T, P)$ は以下の式を満たす.

$$\lim_{m\to\infty} r_m(T_m^*, P) = H(P) \tag{3}$$

上の式は $m \to \infty$（内部ノードの数を無限に増やす）としたとき確率分布 P をもつ定常無記憶情報源 X に対し, タンストール木の平均符号化レートは $H(P)$ に収束することを表している. つまり, タンストール木は内部ノードの数を無限に増やすほどに圧縮率が高くなるのである. しかし, 内部ノードを無限に増やすことは不可能であり, また木が大きくなるということは符号化をする際, 符号器での遅延が大きくなるということを意味しているので本実験においてはハフマン符号のブロック長に合わせた木の大きさ (ノード数をハフマン木とタンストール木で合わせる) で実験を行うものとする. したがって平均符号化レートはノード数が多くなると $H(P)$ に近づくが $H(P)$ とはかなりの差がある.

5　システムにおける平均遅延

5.1　平均遅延の計測

5.1.1　実験内容

タンストール符号を用いた伝送システムの場合, どれほどの遅延が発生するのかを調べた. システムでは出現するシンボルとその到着間隔が確率過程となっているので, 遅延を単なる値として見るために平均遅延を求めシステムの評価をおこなった. 本研究ではハフマン符号を用いた伝送システムとの遅延の比較が最大の目的であるので木の中間ノードの数と情報源の確率を 2 つの符号語で同じにした状態で比較した. システムのシンボルの発生確率を固定した状態で, 木の中間ノードの数を動かして平均遅延を計測した. シンボルの発生確率を 0.01〜0.5 とし, 到着レート λ は λ の上限値の 0.1 倍〜0.95 倍までの間で等間隔の 18 個の点で平均遅延をとった. タンストールの結果を図 3 と図 5, 図 7, 図 9 に示しハフマンの結果を図 4 と図 6, 図 8, 図 10 に示す.

5.1.2　結果と考察

情報源の出現確率が 0.5 のときのタンストール符号の結果の図 3 とハフマン符号の結果図 4 を比較する. 情報源の出現確率が 0.5 のとき, タンストール木とハフマン木は全く同じ形をしている.2 つの符号語の入力のシンボルの長さと出力の符号語の長さが等しくなるので遅延の特性も完全に一致する. 出現確率の偏りが大きくなるほどタンストール符号の遅延は大きくなることがタンストール符号の遅延の図 3(出現確率=0.5) と図 5(出現確率=0.3), 図 7(出現確率=0.1) を比較するとわかる. 図 7(出現確率=0.1) と図 9(出現確率=0.01) を比較すると出現確率が 0.01 のときのほうが遅延が小さくなっているように見えるが, これは到着レートを正規化して表しているからである. 図 11(タンストール符号のノード数 31 の場合の出現確率 0.1 と 0.01 の比較) の正規化していない到着レートで比較すると, 到着レートが 1.6[シンボル/秒] 以上の領域で出現確率 0.1 の遅延が大きくなっているのは, 伝送システムが許容できる到着レートの最大値に近づくためバッファでの待ち時間が急激に増加するからであると考えられるので遅延の特性としては出現確率の偏りが大きいほど遅延は大きくなるといえる. 次にハフマン符号の遅延についてハフマン符号の遅延の図 4(出現確率=0.5) と図 6(出現確率=0.3), 図 8(出現確率=0.1), 図 10(出現確率=0.01) について比較する. ハフマン符号は出現確率の偏りが大きくなるほどに遅延が小さくなっていることが図よりわかる. この理由はブロック長は一定なので符号器での待ち時間はどの確率でも一定であるが, 出現確率の偏りが大きくなるほどに出力される符号語の長さが短くなるため送信時間が短縮されるからである. 以上のことから出現確率の偏りが大きくなるほどタンストール符号とハフマン符号で遅延の差が大きくなるということ

がわかった. まとめると, 出現確率の偏りが小さいときは 2 つの符号語で遅延の差は小さく, 偏りが大きくなるほどタンストール符号は遅延が大きくなるのに対しハフマン符号は遅延が小さくなるということがわかった.

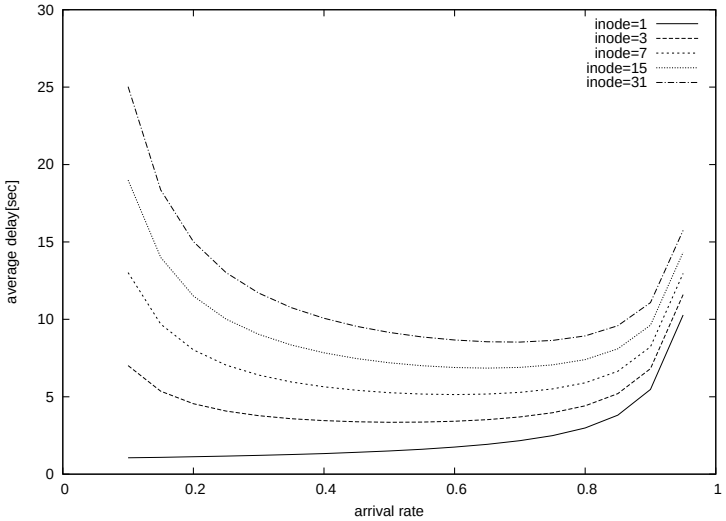

図 3　タンストール符号　平均遅延 p = 0.5 の場合

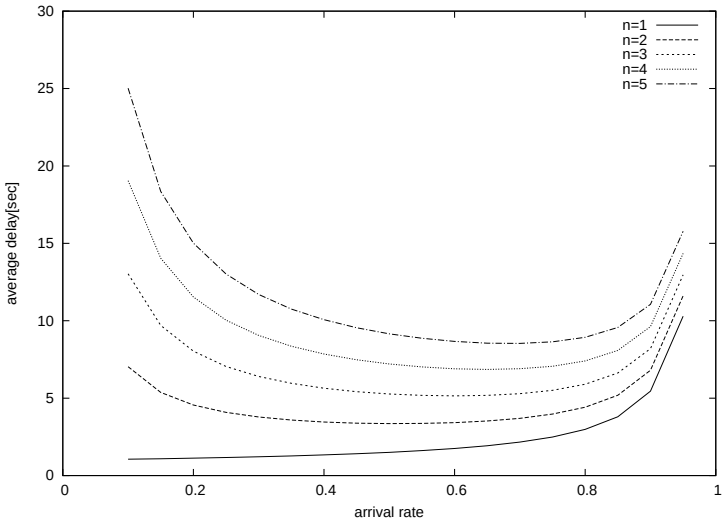

図 4　ハフマン符号　平均遅延 p = 0.5 の場合

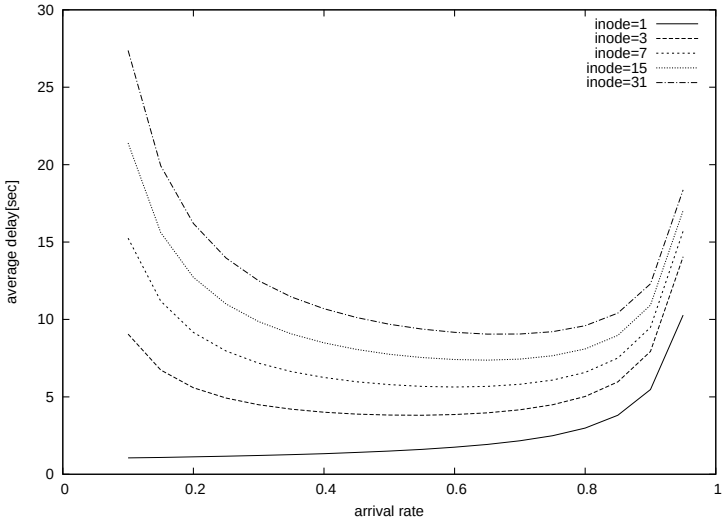

図 5　タンストール符号　平均遅延 $\mathrm{p} = 0.3$ の場合

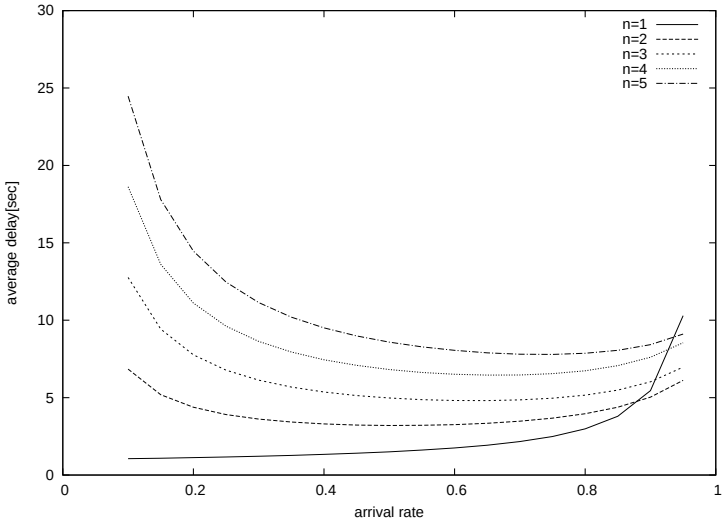

図 6　ハフマン符号　平均遅延 $\mathrm{p} = 0.3$ の場合

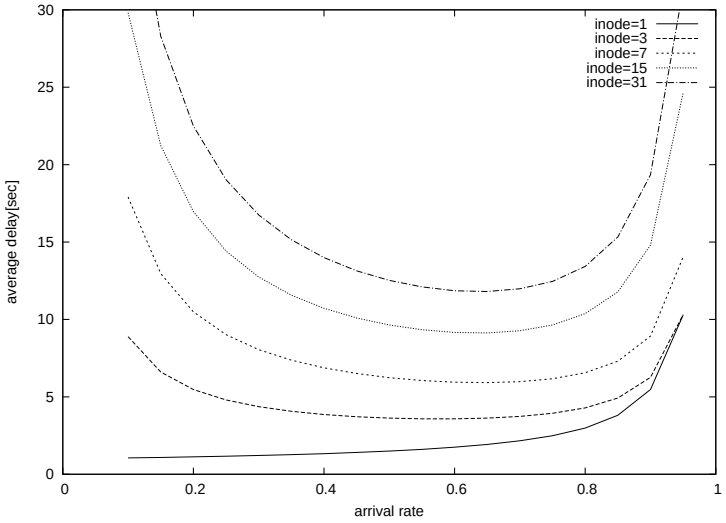

図 7　タンストール符号　平均遅延 $\mathrm{p} = 0.1$ の場合

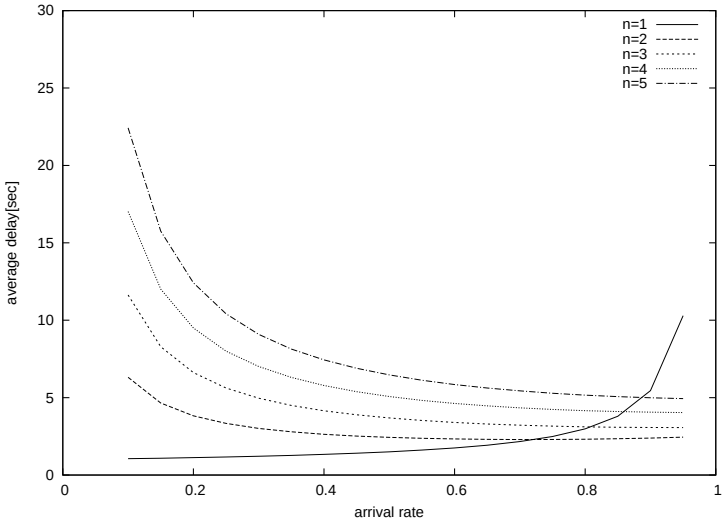

図 8　ハフマン符号　平均遅延 $\mathrm{p} = 0.1$ の場合

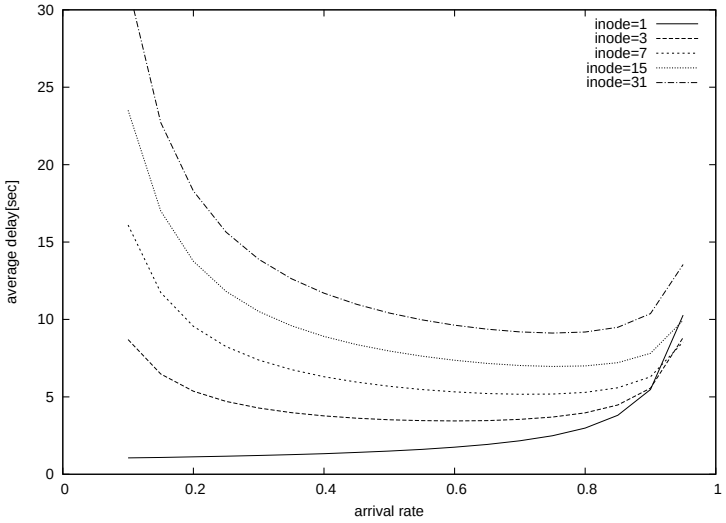

図 9　タンストール符号　平均遅延 p = 0.01 の場合

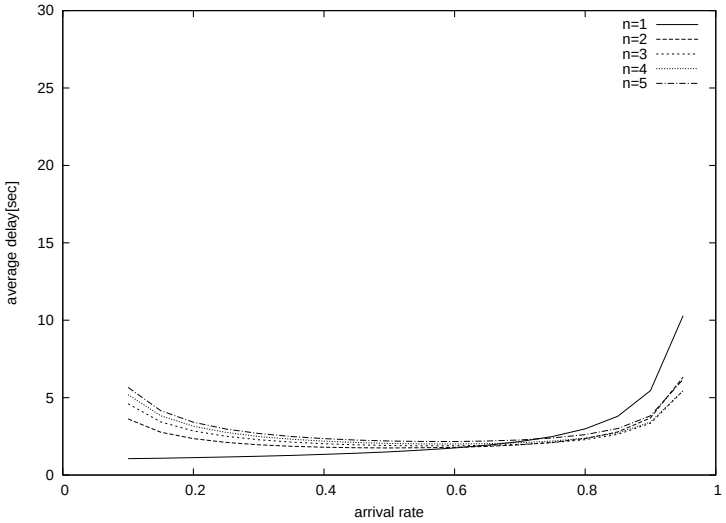

図 10　ハフマン符号　平均遅延 p = 0.01 の場合

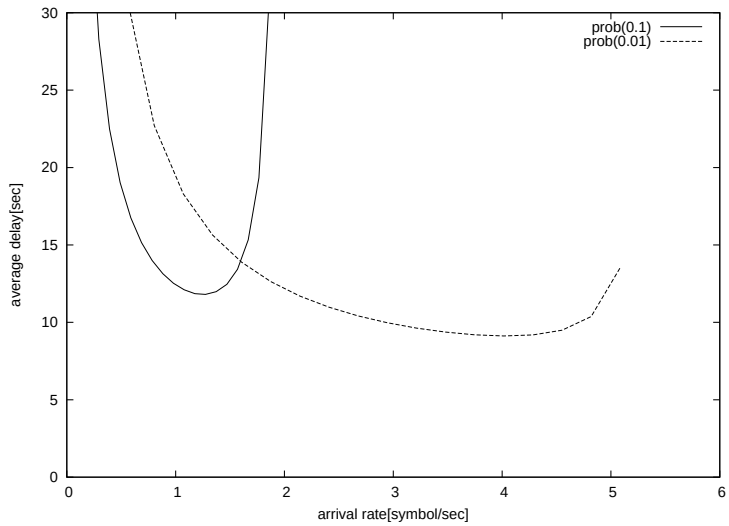

図 11　タンストール符号　平均遅延 31 ノードの p = 0.1 と 0.01 の場合

5.2 平均遅延の内訳

5.2.1 遅延の内訳の導出

本研究の伝送システムで遅延が発生するのは, 符号器とバッファ, 送信機の 3 か所である. これは, ハフマン符号の伝送システムにおいても同様である．ここでは 3 つの遅延の特徴がどのようなものであるのかを調べた. まず, 符号器での待ち時間と送信機での送信時間の理論式を導いた．そして, 上記で得られた平均遅延の値から 2 つの遅延の理論式を引くことで, バッファでの待ち時間を導いた．まず, 送信機での送信時間の理論式であるがタンストール符号は VF 符号なので符号語の長さがすべて一定であるため送信にかかる時間も一定である．符号語長を L とすると,

$$(\text{送信時間}) = \frac{L}{\mu} \tag{4}$$

となる．次に, 符号器での待ち時間は符号器内に入ったある 1 つのシンボルが符号語になるまで平均的に何シンボルの到着を待つことになるのかを求めることによって, 待ち時間の理論式を求めた．まず, ある木を考えた時にあるシンボルが符号器に到着してから符号語となってバッファに送られるまでに待つ到着シンボル数を l とする. また, その木のノードを k とおくとノードに到着する確率を $q_k(k)$ さらに, ノードに到着した元で符号語が完成するまでにシンボルが待たされるシンボル数 l の確率を $q_{l|k}(l|k)$ とすると, 待ちシンボル数の期待値は

$$(\text{待ちシンボルの期待値}) = \sum_{k\geq 0} q_k(k) \sum_{l\geq 0} l \times q_{l|k}(l|k) \tag{5}$$

となるので, 待ち時間の期待値は

$$(\text{待ち時間の期待値}) = \frac{1}{\lambda} \times \sum_{k\geq 0} q_k(k) \sum_{l\geq 0} l \times q_{l|k}(l|k) \tag{6}$$

となる.

実験では $q_k(k)$ を求めるために, 木の各ノードにあらかじめ確率を与えておき図 12 の例の各枝に関する行列計算を繰り返すことにより各ノード k の確率の定常状態を求めることにより k の確率を導き出した.

情報源の出現確率が 0.5 のとき q_k の値は収束せずに周期的に値が変化してしまうので, どんな出現確率の値でも q_k が収束するようにするために (7) 式の行列計算では右辺の $(q_k(1)q_k(2)q_k(3)q_k(4))^T$ と左辺の $(q_k(1)q_k(2)q_k(3)q_k(4))^T$ をそれぞれの行で足して 2 で割ったものを, 新たに右辺に代入して左辺の $(q_k(1)q_k(2)q_k(3)q_k(4))^T$ を求めるといった方法を用

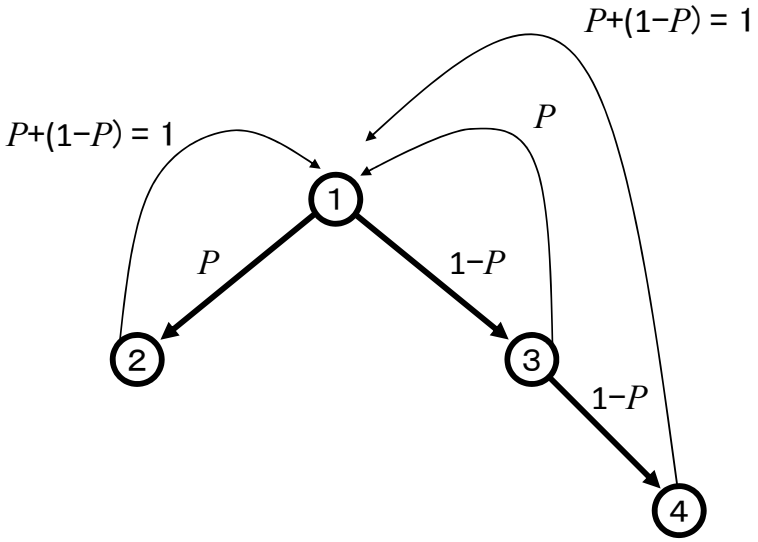

図 12　ノード k の確率 $q_k(k)$

いて q_k を求めている.(T は転置行列を表す)

$$\begin{pmatrix} q_k(1) \\ q_k(2) \\ q_k(3) \\ q_k(4) \end{pmatrix} = \begin{pmatrix} 0 & 1 & P & 1 \\ P & 0 & 0 & 0 \\ 1-P & 0 & 0 & 0 \\ 0 & 0 & 1-P & 0 \end{pmatrix} \begin{pmatrix} q_k(1) \\ q_k(2) \\ q_k(3) \\ q_k(4) \end{pmatrix} \tag{7}$$

5.2.2　平均遅延の内訳の考察

情報源シンボルの出現確率の偏りが大きいほど 2 つの符号語で遅延の差が大きくなり考察がしやすいので出現確率が 0.01 の場合のタンストール符号の遅延の内訳図 9 とハフマン符号の遅延の内訳図 10 について比較する. 縦軸の範囲が 2 つの図で異なるが, バッファでの待ち時間は 2 つの図であまり違いがなく, 遅延の差の原因は待ち時間と送信時間にあることが図より分かる. ここで, タンストール符号のほうが待ち時間と送信時間が長くなる原因を図 13 より説明する. 情報源シンボルの出現確率が 0.01 のときタンストール木もハフマン木も図 13 の木の形をしている.(図 13 の例ではスペースの関係上ノードの数を 7 とした) このとき, ハフマン符号はシンボル bbb が出現しやすいので符号語は 0 だけが出力されやすい. つまり入力 3 シンボル出力 1 ビットである. それに対してタンストール符号はシンボル b の出現確率が 99 パーセントなので符号語 111 が出力されやすい. つまり入力 7 シンボル出力 3 ビットとなる. 以上の理由から, タンストール符号のほうが入力長, 出力長共にハフマン符号よりも長いことがわかる. 入力長が長いということは符号器での待ち時間が長いということであり, 出力長が長いという

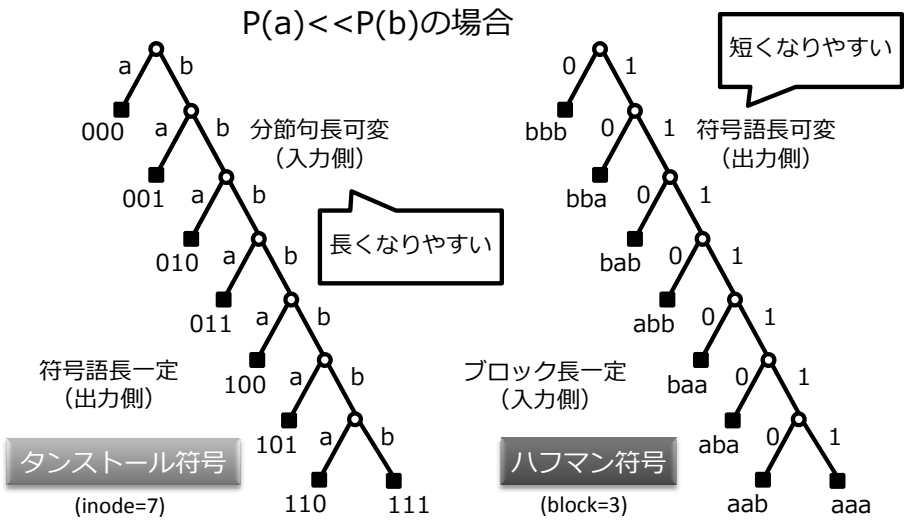

図 13　p = 0.01 のときのタンストール木とハフマン木

ことは送信機での送信時間が長くなるということを意味している. タンストール符号の符号語長は一定であるため, 確率の偏りにより長くなるのは分節句長である．タンストール符号の待ち時間が先に述べた理由により大きくなったこととハフマン符号の符号語長が短くなったことが, ハフマン符号に比べタンストール符号の遅延が大きくなった原因であると考えられる.

6　まとめ

タンストール符号を用いた伝送システムを構築してシステムの平均遅延を計測しハフマン符号を用いたシステムと到着レートに対する遅延の特性を比較する研究をおこなった. 実験の結果はタンストール符号を用いたシステムでは情報源シンボルの発生確率に偏りがない場合はハフマン符号のシステムと遅延の特性に大きな違いがないが, 発生確率の偏りが大きくなるとハフマン符号は遅延が小さくなるが, タンストール符号は遅延が小さくならず発生確率の偏りによる遅延の変化はほとんど見られなかった．この原因について考えると, タンストール符号もハフマン符号同様シンボルの発生確率の偏りが大きいと平均符号化レートが小さくなり送信にかかる時間はタンストール符号も小さくなるが, ハフマン符号のブロック長が一定なのに対し, タンストール符号の分節句長は可変長でありシンボルに偏りがある程, 分節句は長くなる性質があるので符号器での待ち時間が長くなってしまうからである．結果的にタンストール符号はハフマン符号に比べて遅延が大きいということがわかったが, 現実的な情報源には本研究の実験で用いたような情報源シンボルの出現確率が 0.01 といったものは少ないので, 現実的な伝送

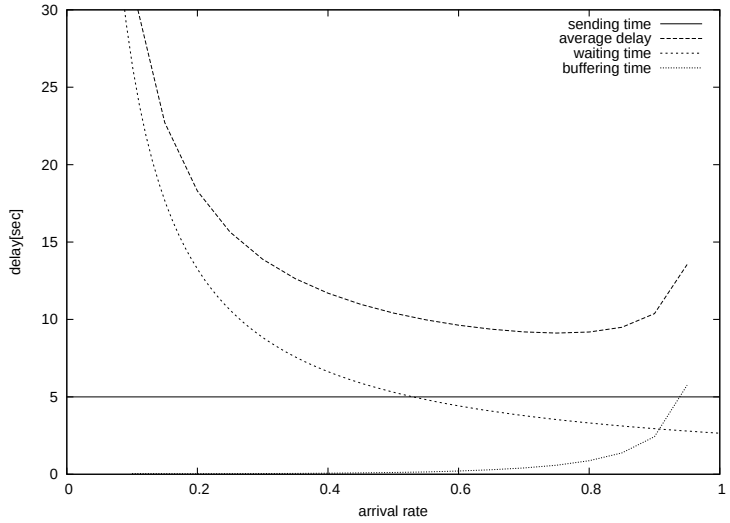

図 14　タンストール符号　平均遅延 p = 0.01 の場合

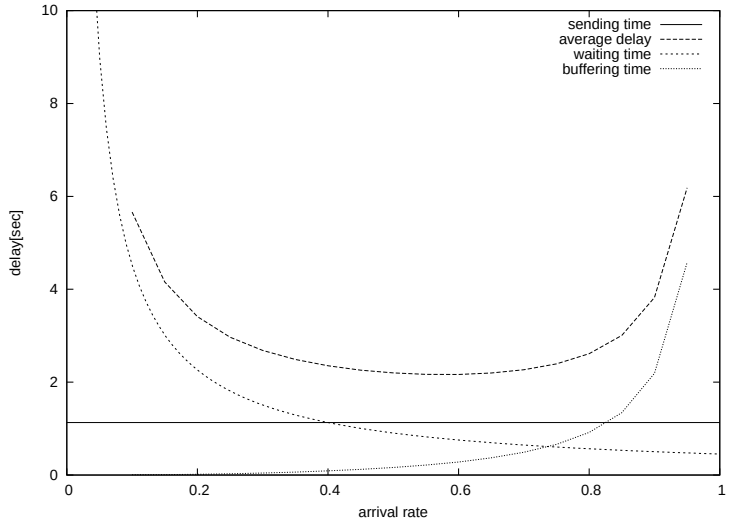

図 15　ハフマン符号　平均遅延 p = 0.01 の場合

システムで用いた場合のハフマン符号との遅延の差はそれほど大きくないと予想される. したがって, 移動体通信などのようにノイズの影響を受けやすい伝送システムではタンストール符号を用いたほうがメリットが大きい場合もあるのではないかと考えられる.

謝辞

本研究を終えるにあたり, 終始一貫して丁寧なご指導を賜わりました指導教員の西新幹彦先生に心より感謝と敬意の意を申し上げます.

参考文献

[1] 小林欣吾, 森田啓義, 情報理論講義, 培風館, 2008.

[2] 杉原辰徳「算術符号を用いた伝送システムにおけるポアソン到着シンボルの遅延」信州大学工学部, 学士論文,2008.

[3] 小泉太生樹「ハフマン符号を用いた伝送システムにおけるポアソン到着のシンボルの遅延」信州大学工学部, 学士論文,2009.

付録 A　補題の証明

タンストール符号の平均符号化レートが $m \to \infty$ としたときに情報源のエントロピー $H(P)$ に近づくことを証明する.

タンストール符号の符号語長は全ての符号語で等しい長さなので, 葉の枚数が L の木の符号語長は

$$(\text{葉の枚数が } L \text{ のタンストール符号の符号語長}) = \lceil \log L \rceil \tag{8}$$

(log の底は符号語アルファベットの大きさ)
で表すことができるので, ノード数が m の T の葉の総数を L_m とし T の平均分節句長を $\bar{l}_T$ とすると平均符号化レート $r_m(T,P)$ は下の式で定義される.

$$r_m(T,P) = \frac{\lceil \log L_m \rceil}{\bar{l}_T} \tag{9}$$

また, 分節木 T に対して以下の式が成り立つ

$$H(T) = \bar{l}_T H(P) \tag{10}$$

(8)(9) 式より, 以下の式が得られる.

$$r_m(T,P) = \frac{\lceil \log L_m \rceil \cdot H(P)}{H(T)} \tag{11}$$

分節木 T に対して以下のようにおく,

$$P^{\max}(T) = \max_{\nu \in \mathcal{L}_T} P(\nu) \tag{12}$$

$$P^{\min}(T) = \min_{\nu \in \mathcal{L}_T} P(\nu) \tag{13}$$

$$P^{\min}(T) = \min\{P(x), x \in \mathcal{X}\} \tag{14}$$

(ただし,$\mathcal{L}_T$ は T の葉の集合)
このとき次式が成り立つ

$$\frac{P^{\max}(T^*)}{P^{\min}(T^*)} \leq \frac{1}{P^{\min}} \tag{15}$$

(T^* は Tunstall 木の構成法によって作られた木)
$H(T)$ は $\mathcal{L}_T$ のエントロピーであり, 以下のように表すことができる.

$$H(T) \leq \log L_m \tag{16}$$

さらに,

$$H(T) = -\sum_{\nu \in \mathcal{L}_T} P(\nu) \log P(\nu) \tag{17}$$

に対して任意の $\nu \in \mathcal{L}_T$ に成り立つ不等式

$$-\log P(\nu) \geq -\log P^{\max}(T) \tag{18}$$

を代入すると

$$H(T) \geq -\sum_{\nu \in \mathcal{L}_T} P(\nu) \log P^{\max}(T) \tag{19}$$

$$= -\log P^{\max}(T) \sum_{\nu \in \mathcal{L}_T} P(\nu) \tag{20}$$

$$= -\log P^{\max}(T) \tag{21}$$

を得る. ここで T としてタンストール木を考えると (14) 式より (20) 式は

$$H(T_m^*) \geq -\log P^{\min}(T_m^*) + \log P^{\min} \tag{22}$$

と変形することができる. また, 任意の $\nu \in \mathcal{L}_T$ に対して常に

$$P^{\min}(T) \leq \frac{1}{L_m} \tag{23}$$

が成り立つので, これを (21) 式に代入すると

$$H(T_m^*) \geq \log L_m + \log P^{\min} \tag{24}$$

を得る.(15) 式と (23) 式をまとめると

$$\log L_m + \log P^{\min} \leq H(T_m^*) \leq \log L_m \tag{25}$$

となる. (24) 式と実数 t に対して $t \leq \lceil t \rceil < t + 1$ が成り立つことから符号化レート (10) 式は

$$H(P) \leq r_m(T_m^*, P) \leq \frac{\log L_m + 1}{\log L_m + \log P^{\min}} H(P) \tag{26}$$

と評価できる.$m \to \infty$ のとき $L_m \to \infty$ なので (25) 式の右辺の係数は

$$1 + \frac{1 + \frac{1}{\log L_m}}{1 + \frac{\log P^{\min}}{\log L_m}} = \frac{1 + 0}{1 + 0} = 1 \tag{27}$$

となり 1 に収束する. つまり $m \to \infty$ としたとき以下の式が成り立つ.

$$\lim_{m \to \infty} r_m(T_m^*, P) = H(P) \tag{28}$$

(証明終)

付録 B　ソースコード

B.1　指数乱数を発生させるプログラム

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include "arithmetic_coding.h"

/*****************************************指数分布乱数を発生************************************************/
#define MRND 1000000000L
static int jrand;
static long ia[56];

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();
}

double ramda;  /* 到着レートλ；λが大きくなるほど，到着間隔は小さくなる．*/
double ramda_x;

double rand_(void)
{
    return (-(1/ramda)*log(1-(rnd())));  /* 指数分布 */
```

```
}

/*****************************************************************************************************/
double present_time;  /* 現在時刻 */
double arrival_time;  /* 到着時刻 */
double sending_time;  /* 送信終了時刻 */

#define ARRIVING_LIST_MAX 1000
#define SENDING_BUFFER_LIST_MAX 1000

double arriving_list[ARRIVING_LIST_MAX];
int arriving_count=0;
int arriving_in=0;
int arriving_out=0;

int sending_buffer_list[SENDING_BUFFER_LIST_MAX];
int sending_buffer_count=0;
int sending_buffer_in=0;
int sending_buffer_out=0;
```

B.2 伝送システムのプログラム

```
/*************************************バッファ(エンコーダ側)*****************************************/
void add_arriving_list(double time)
{
    if (arriving_count >= ARRIVING_LIST_MAX){

        printf("warning1!!");
        exit(0);
    }
    arriving_list[arriving_in] = time;
    arriving_in = (arriving_in + 1) % ARRIVING_LIST_MAX;
    arriving_count++;
    return;
}

double get_arriving_list()
{
    double time;

    if (arriving_count == 0){

        printf("warning2!!");
        exit(0);
    }
    time = arriving_list[arriving_out];
    arriving_out = (arriving_out + 1) % ARRIVING_LIST_MAX;
    arriving_count--;
    return(time);
}

/*****************************************バッファ（送信器側)*****************************************/
void add_sending_buffer_list(int id)
{
    if (sending_buffer_count >= SENDING_BUFFER_LIST_MAX){

        printf("warning3!!");
        exit(0);
    }
    sending_buffer_list[sending_buffer_in] = id;
    sending_buffer_in = (sending_buffer_in + 1) % SENDING_BUFFER_LIST_MAX;
    sending_buffer_count++;
    return;
}
```

```
int get_sending_buffer_list()
{
    int id;


    if(sending_buffer_count == 0){

        printf("warning4!!");
        exit(0);
    }
    id=sending_buffer_list[sending_buffer_out];
    sending_buffer_out=(sending_buffer_out+1)%SENDING_BUFFER_LIST_MAX;
    sending_buffer_count--;
    return(id);
}

/*****************************************Tunstall 符号化*************************************************/
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdlib.h>

struct branch {  /* 構造体で branch を作成 */
    double p;  /* 確率 */
    struct node * n;  /* ポインタ */
    int id;  /* 葉の番号 */
};

struct node {  /* 構造体で node を作成 */
    struct node *parent;
    struct branch b0;
    struct branch b1;
};

int internalnode;   /* 中間接点の数 */
double send_time;

double prob0;
double prob1;

#define PROB0 0.1  /* 左側のノードの確率 */
#define PROB1 (1 - PROB0)  /* 右側のノードの確率 */
struct node *inode;
struct node **leaf;

#define INPUT_LENGTH (1 << 20)
int symbol_count;
double delay;
double sum_delay;

/*****************************************************************************************************/
void
make_tree(double prob0)     /* TunstallTree */
{
    int m;

    inode[0].parent = NULL;
    inode[0].b0.p = prob0;
    inode[0].b1.p = 1.0 - prob0;
    inode[0].b0.n = NULL;
    inode[0].b1.n = NULL;

    for (m = 1; m < internalnode; m++){
        int i;
        double pmax = 0;
        struct branch *sprout;
        struct node *parent;
```

```
        for (i = 0; i < m; i++){
            if (inode[i].b0.n == NULL && inode[i].b0.p > pmax){
                pmax = inode[i].b0.p;
                sprout = &(inode[i].b0);
                parent = &(inode[i]);
            }
            if (inode[i].b1.n == NULL && inode[i].b1.p > pmax){
                pmax = inode[i].b1.p;
                sprout = &(inode[i].b1);
                parent = &(inode[i]);
            }
        }

        sprout->n = inode + m;  /* (*sprout).n = &(inode[m]); */
        inode[m].parent = parent;
        inode[m].b0.p = sprout->p * prob0;
        inode[m].b1.p = sprout->p * (1.0 - prob0);
        inode[m].b0.n = NULL;
        inode[m].b1.n = NULL;

    }
    return;
}

double ave_input_length(void)
{
    int m=0;
    double sum_prob=0;

    for (m = 0; m < internalnode; m++){
    sum_prob += inode[m].b0.p + inode[m].b1.p;  /* sum_prob = sum_prob + (inode[m].b0.p + inode[m].b1.p); */
    }
    return(sum_prob);
}

int
enum_tree_rec(struct node *node, int seq)
{
    if (node->b0.n == NULL){
        node->b0.id = seq++;
        leaf[node->b0.id] = node;
    }else{
        seq = enum_tree_rec(node->b0.n, seq);
    }
    if (node->b1.n == NULL){
        node->b1.id = seq++;
        leaf[node->b1.id] = node;
    }else{
        seq = enum_tree_rec(node->b1.n, seq);
    }
    return(seq);
}

int
enum_tree()
{
    return(enum_tree_rec(inode, 0));
}

/*****************************************Tunstall エンコーダ*********************************************/
struct node *node_encode;

void
encode(char bit)
{
    if (bit == '0'){
        if (node_encode->b0.n == NULL){
            add_sending_buffer_list(node_encode->b0.id);
            if (sending_time < 0.0){
```

```
                sending_time = present_time + send_time;
            }
            node_encode = inode;
        } else {
            node_encode = node_encode->b0.n;
        }
    } else {
        if (node_encode->b1.n == NULL){

            add_sending_buffer_list(node_encode->b1.id);
            if (sending_time < 0.0){
                sending_time = present_time + send_time;
            }
            node_encode = inode;
        } else {
            node_encode = node_encode->b1.n;
        }
    }
    return;
}

/*************************************Tunstall デコーダ (delay も出力)*************************************/
void
decode_rec(struct node *node, struct node *child)
{
    if (node->b0.n == child){
        if (node->parent != NULL){
            decode_rec(node->parent, node);
        }
        delay = (present_time - get_arriving_list());
        printf("%f\n",delay);     /* delay 表示 */
        sum_delay = sum_delay + (delay);
    } else {
        if (node->parent != NULL){
            decode_rec(node->parent, node);
        }
        delay = (present_time - get_arriving_list());
        printf("%f\n",delay);     /* delay 表示 */
        sum_delay = sum_delay + (delay);
    }
    return;
}

void
decode(int id)
{
    if (leaf[id]->parent != NULL){
        decode_rec(leaf[id]->parent, leaf[id]);
    }
        delay = (present_time - get_arriving_list());
        printf("%f\n",delay);     /* delay 表示 */
        sum_delay = sum_delay + (delay);
    return;
}

/*************************************************************************************************************/
void
sending_finish()     /* 送信終了処理 */
{
    int id;

    id = get_sending_buffer_list();
    decode(id);     /*デコード処理*/
    if(sending_buffer_count == 0){
        sending_time = -1.0;
    } else {
        sending_time = present_time + send_time;
    }
    return;
```

```
}

/*****************************************************************************************************/
int
generate_symbol()
{
    int x;
    if ( rnd() <= prob0 ){
        x = '0';
    } else {
        x = '1';
    }
    return(x);
}

void
arrival_symbol()     / *到着シンボル処理 */
{
    add_arriving_list(present_time);
    encode(generate_symbol());     / *エンコード処理 */
    symbol_count++;
    if (symbol_count >= INPUT_LENGTH){
        arrival_time = -1.0;
    } else {
        arrival_time = present_time + rand_();  /* 次の到着予定時刻（現在時刻 + 指数分布秒） */
    }

    return;
}

double
make_send_time(int inode)
{
    int n;

    for (n = 0; inode != 0; n++){
        inode >>= 1;     /* 右へ 1 つビットシフト  inode = inode >> 1 */
    }
    return(n);
}

int
main(int argc,char **argv[])
{
    if (argc != 4){
        printf("no prob0 (0.1〜0.5)\n");
        printf("no internalnode (1〜2^32)\n");
        printf("no ramda_x (0.1〜0.9)\n");
        exit(1);
    }
    prob0 = atof(argv[1]);
    prob1 = 1 - prob0;
    internalnode = atoi(argv[2]);
    ramda_x = atof(argv[3]);
    send_time = make_send_time(internalnode);
    inode = (struct node *)calloc(internalnode, sizeof(struct node));
    if (inode == NULL){
        printf("error on %d\n", __LINE__);
        exit(1);
    }
    leaf = (struct node **)calloc(internalnode + 1, sizeof(struct node *));
    if (leaf == NULL){
        printf("error on %d\n", __LINE__);
        exit(1);
    }

    make_tree(prob0);  /* TunstallTree を作る */
    ramda = ramda_x/(send_time/ave_input_length());
    init_rnd(12345);     /* 任意の long で初期化．省略不可．*/
```

```
    enum_tree();          /* TunstallTree に id を付ける */
    present_time = 0.0;  /* 現在時刻 */
    arrival_time = rand_();  /* 到着時刻 */
    sending_time = -1.0;   /* 送信終了時刻 */
    symbol_count = 0;
    node_encode = inode;  /* ポインタ (node_encode) をルート (inode) に戻す */
    delay = 0.0;
    sum_delay = 0.0;

    while ((arrival_time >= 0.0) || (sending_time >= 0.0)){

        if (arrival_time<0 || (sending_time>=0 && arrival_time>sending_time)){

            present_time = sending_time;
            sending_finish();  /* 送信終了処理 */
        } else {
            present_time = arrival_time;
            arrival_symbol();  /* 到着シンボル処理 */
        }
    }
    return(0);
}
```