信州大学
大学院工学系研究科

修士論文

遅延特性改善のための
分節木と符号語に関する実験的考察

指導教員　西新 幹彦 准教授

専攻　電気電子工学専攻
学籍番号　10TA205A
氏名　泉 陽一

2012 年 3 月 14 日

目次

1 序論

携帯電話における緊急地震速報などのように, データを早く伝送したい場合, どのように伝送すべきかが問題となる. 一般的にデータサイズを小さくする目的で符号化をするといったことはよく行われているが, 不規則に到着するシンボルに対して遅延を小さくする目的でデータを符号化するといったことはあまり行われていない. しかし, 緊急地震速報などへの応用が考えられ, このような研究は工学的に重要な意味を持っている.

従来研究では, 算術符号とハフマン符号, タンストール符号を用いた研究があったが [2][4][5], 本研究では, ハフマン符号やタンストール符号のように木で符号化が出来る符号を一般に考え, ハフマン符号やタンストール符号よりも遅延特性に優れる符号を構成することを考える. 具体的には,FV 符号 (variable-length-to-fixed-length code) であるハフマン符号と VF 符号 (variable-length-to-fixed-length code) であるタンストール符号に対して, FV 符号や VF 符号よりも広い範囲の符号を持つ VV 符号 (variable-length-to-fixed-length code) の中から遅延がより小さくなる符号を見つけることが本研究の目的である.

2 研究の背景

この章では, 本研究と従来研究 [2][3][4][5] の枠組みについて説明する.

2.1 不規則に到着するシンボルの符号化

本研究と従来研究の枠組みは, 図 1 に示す伝送システムにおいて情報源シンボルの伝送遅延を求めることによって, 用いた符号の遅延特性を評価することである. 特に本研究では, 評価だけでなく遅延特性に優れる符号を構成することを目的としている. 本研究と従来研究で用いる, 図 1 の伝送システムでは, 情報源シンボル (a または b) は到着レート λ[シンボル/秒] のポアソン到着で符号器に到着する. 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. そして, 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送信バッファで待たされることになる. 送信機から復号器までは符号語は送信レート μ[ビット/秒] で送られる. 到着レート λ と送信レート μ は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, 本研究では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.

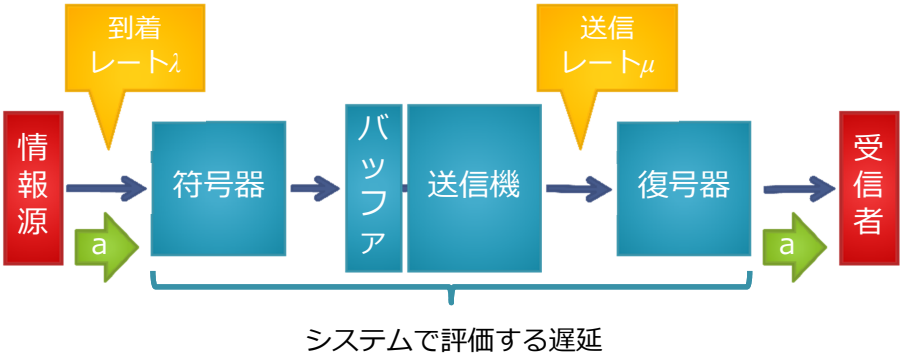

図 1　伝送システム

2.2　符号化に起因する遅延

ここでは, 図 1 の伝送システムにおける遅延を定義する. 評価を行う遅延は, 図 1 のシステムにシンボルが入力されてから出力されるまでの時間であるが, 具体的には, 符号器での「符号化時間」とバッファにおける「待ち時間」, 送信機で符号語送信にかかる「送信時間」の 3 つの時間の合計が遅延である. このとき, 情報源から符号器までの伝送時間と復号化に掛かる時間, 復号器から受信者までの伝送時間はシステムの物理的な能力に依存する部分であり, 符号の理論的な遅延特性を評価する上で本質的な部分ではないので, 無視するものとする.

2.3　従来研究とフィードバック

研究の枠組みで説明した図 1 の伝送システムの符号器・復号器として, 算術符号, ハフマン符号, タンストール符号を用いて, 伝送遅延を評価した従来研究がある [2][4][5]. 各符号に対する研究により, いずれもシンボルの出現確率の偏りが大きく到着レートが高い領域においては, 情報源圧縮により符号語の送信時間が短縮されるため遅延を改善できるが, 到着レートが低い領域では, 符号化に掛かる時間の影響により遅延が大きくなることが分かっている. これは, 到着レートが低いときは通信路容量に余裕があるにもかかわらず, シンボルは, システムに到着してから送信状態になるまでの間に, 符号器の内部で無駄に待たなければならないからである. これを改善するためには, 通信路が遊休状態になった瞬間に, すぐに次の符号語の送信を開始すれば良いと考えられる. そこで, 送信機が遊休状態であることを符号器に伝える仕組みを考え, この仕組みをフィードバックと呼ぶ. 算術符号器に対して送信機よりフィードバック信号を送ることで符号化に掛かる時間を短縮し, 遅延を改善するといったことが従来研究 [4] で行われている. しかし, フィードバック機能の真価を知るためには, さらにいろいろな符号を使っ

て調べる必要がある．

3　分節木と符号語の探索法

本研究の目的は, フィードバック機能がある伝送システムにおいて遅延を小さくする符号とは, どのようなものであるのかを求めることである (図 2)．遅延を小さくする符号を構成するためには, 遅延を小さくするという意味でより良い分節木の形を決めなければならない．さらに, フィードバックを可能にするために分節木の中間ノードにも符号語を割り当てる必要がある．しかし, 符号語数の増加に伴い平均符号語長も長くなるので, 符号語を割り当てる中間ノードと割り当てた場合の符号語長を考える必要がある．最適な分節木を定式的に求めることは, フィードバック機能の影響により非常に困難であるので, 本研究ではよりよい分節木を実験的に求めることにした．この章では, 本研究におけるフィードバックの実現方法の詳細と良い符号の探索法について説明する．

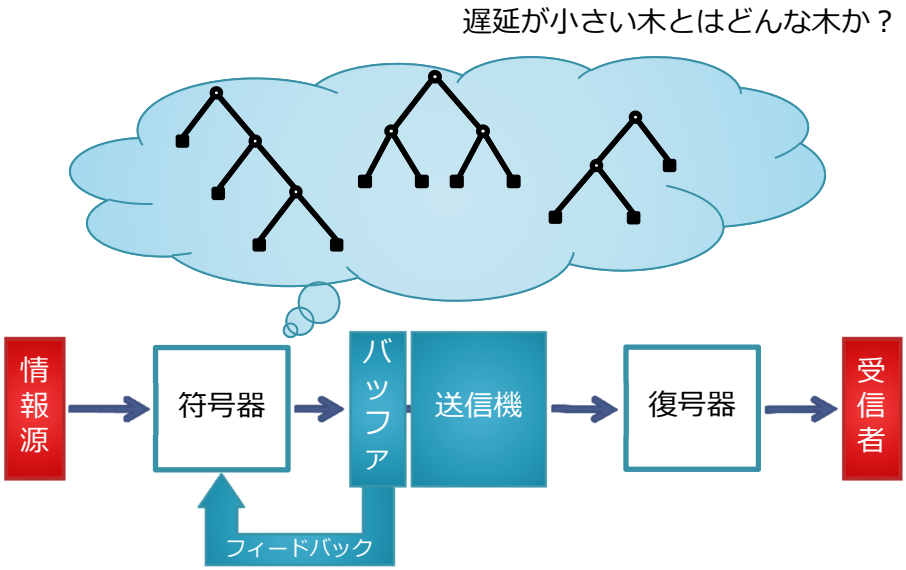

図 2　研究の概要

3.1　分節木とフィードバック

算術符号にフィードバックを用いた研究 [4] が行われていたことは 2 章で述べたが, ハフマン符号 [3] とタンストール符号 [5] については, 到着レートが低い領域では遅延が大きくなることは判明していたが, フィードバックを用いて符号化にかかる時間を短縮するといったこと

は行われてこなかった．そこで, 本研究の予備実験としてハフマン符号とタンストール符号にフィードバックを適用したところ, 遅延の改善が見られた．具体的には, 図 3 に示す分節木を用いてハフマン符号やタンストール符号を表現し, この分節木の中間ノードに符号語を割り当てることによってフィードバックを可能にした．しかし, フィードバックを適用できるように符号語を割り当てると元の符号とは違う符号になってしまう．言い換えると, フィードバックを適用した時点でハフマン符号もタンストール符号も別の符号になる．本研究の目的は, 特定の符号にこだわることではなく, 純粋に伝送遅延を小さくするためにはどのように符号化を行うべきかを探ることにある．そこで, ハフマン符号やタンストール符号などの既存の符号に限定せず, さらに, フィードバックを用いることにより, より広い符号の範囲から遅延特性に優れる符号を見つけ出すことを考える．

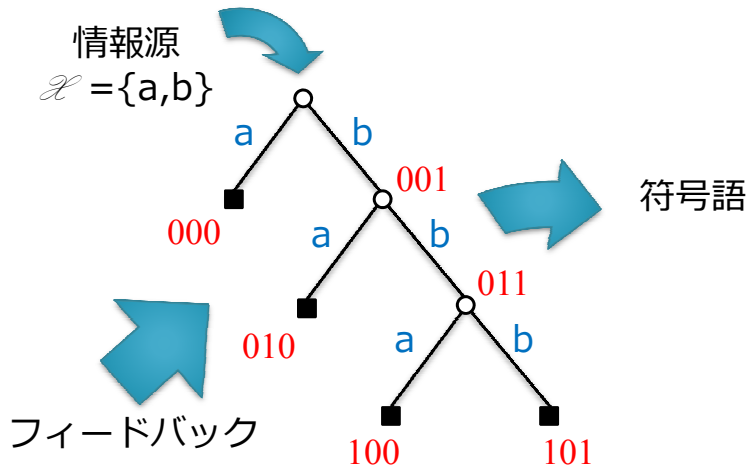

図 3　フィードバックが可能な分節木

3.2　遅延を小さくする分節木を求めるためのアプローチ

遅延を小さくする分節木を求めるための本研究のアプローチは次の通りである（図 4）．最初に適当な初期値を用いて伝送実験を行い, そこから得られるデータを元にして分節木の符号語の割り当て直しを行う．新しい符号語を割り当てられた分節木を用いて再び伝送実験を行い, 符号語の割り当て直しを行うということを, 伝送実験により得られる遅延が収束するまで行う．収束したところで, 今度は分節木の形を実験データにより変形し, また伝送実験を行う．符号語の割り当て直しと分節木の変形の方法は以降の節で具体的に説明する．

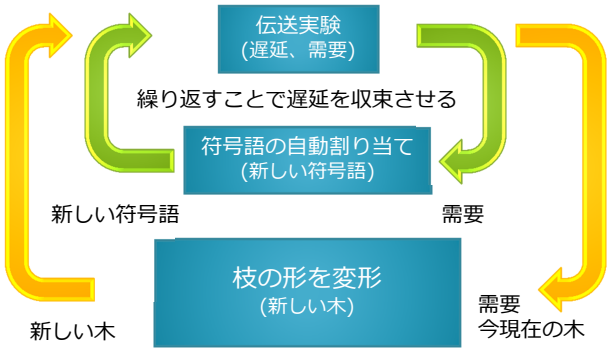

図 4　最適な分節木を求めるためのアプローチ

3.3　符号語の割り当てと需要

分節木に付いている符号語の割り当て直しは, 分節木 (図 3) の内部ノードと葉を重み付けし, ハフマン符号化により各内部ノードと葉に符号語を割り当てることにより行なわれる (図 5). 内部ノードと葉の重み付けを行う際, 図 3 の分節木において単純に符号語を出力した葉や中間ノードをカウントした頻度ではなく, 需要というものを用いる. この需要とは, 到着したシンボルが分節木をルートノードから新たに到着するシンボルに従って枝を辿るとき, フィードバック信号が送られた瞬間にシンボルが存在する内部ノードとフィードバック信号が送られていないときに符号語を出力した葉をカウントしたものである. 需要を用いることにより, 分節木において本当に必要とされるノードを知ることができるようになる. また, 次の節で説明する分節木の変形においても需要を用いる.

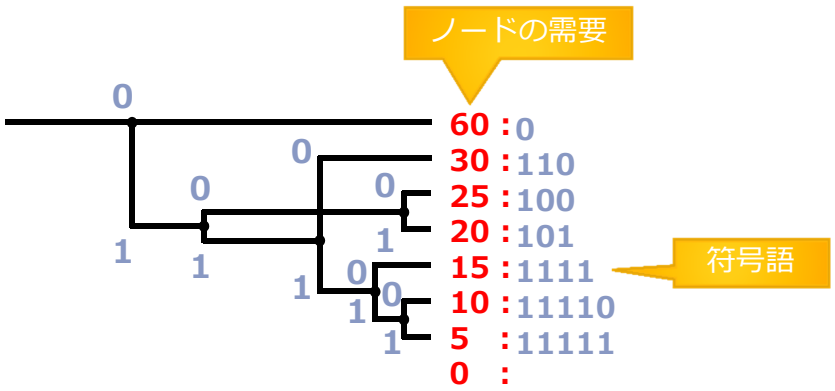

図 5　ハフマン符号による符号語の割り当て直し

3.4 分節木の変形

分節木の変形について説明する. 遅延が小さくなるように分節木を変形しようと考えたとき, 木の深さはシンボルの到着レート, 枝の偏りはシンボルの出現確率の影響を受けることは容易に想像できる. しかし, 具体的にどのような形が最良かは分からない. そこで, 前節で説明した伝送実験により得られる需要を用いて, 遅延がより小さくなる分節木を探索する方法を提案する. また, 闇雲に分節木を拡大しても遅延が大きくなるだけであるので, 予め最大の符号語数を決めておき, その元で分節木を変形することにした. 以上のことから, 分節木の変形は図 6 のような手順を用いる.

手順 1　深さ 1 のツリーを用意する.

手順 2　伝送実験を行い, 各ノードと葉に対する需要を得る.

手順 3　最大の符号語数 $= 3$ の場合

最大の需要を持つ葉を中間ノードに変えて, さらに符号語を取り外す. 変形終了.

手順 4　最大の符号語数 $\geq$ 現在の符号語数 $+2$ の場合

ツリーの葉の中で最大の需要を持つものを中間ノードとして, その下に枝を伸ばす.

手順 5　最大の符号語数 $=$ 現在の符号語数 $+1$ の場合

符号語が割り当てられている中間ノードの中で最小の需要を持つものから符号語を取り外す. そして, 葉の中で最大の需要を持つものを中間ノードとして, その下に枝を伸ばす. 手順 2 を行く.

手順 6　最大の符号語数 $=$ 現在の符号語数 かつ 符号語をもつ中間ノード数 ≥ 2 の場合

符号語が割り当てられている中間ノードの中で需要が小さいものを二つ選び符号語を取り外す. そして, 葉の中で最大の需要を持つものを中間ノードとして, その下に枝を伸ばす.

符号語をもつ中間ノード数 $= 0$ ならば変形を終了し, そうでなければ手順 2 を行う.

手順 7　最大の符号語数 $=$ 現在の符号語数 かつ 符号語をもつ中間ノード数 $= 1$ の場合

符号語が割り当てられている中間ノードの中で最小の需要となるものから符号語を取り外す. さらに, 最大の需要を持つ葉から符号語を取り外すとともに葉を中間ノードして枝を伸ばして, 変形を終了する.

図 6　分節木の変形方法

4　実験結果と考察

この章では, 提案法により得られた符号と従来研究で用いた符号とで遅延の比較を行う．比較の際, 各符号において, 与えられた到着レートに対して最も遅延特性に優れる分節木を用いた．これは, 純粋に伝送遅延の大きさを比較するためである．用いる情報源はシンボル a または b を出力する二元情報源とし, シンボルの出現確率を固定して到着レートを変化させて実験を行った．まず, 到着レートに対する各符号の遅延特性を示して考察を行い．次に, 送信レート（伝送レート）に対する遅延特性を評価する．

4.1　到着レートに対する伝送遅延

まず始めに, シンボルの出現確率が 0.5 の場合の遅延特性を図 7 に示す．結果は図 7 に示す

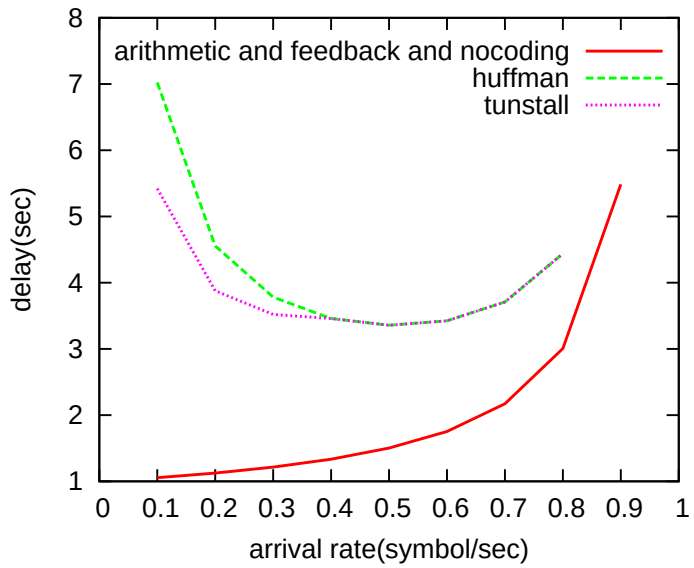

図 7　prob0=0.5 のときの各符号の遅延特性

ように, 無圧縮（nocoding）と提案法により構成した符号（feedback）, 算術符号 (arithmetic) の遅延が一致した．ハフマン符号（huffman）とタンストール符号（tunstall）はほぼ一致しているが, 到着レートが 0.4 以下の領域で若干タンストール符号のほうが遅延が小さくなることが分かった．確率が 0.5 では圧縮は不可能なので, 提案法で木を深くしても遅延が大きくなるだけである．そこで, 提案法の分節木は深さ 1 となり実質的に無圧縮となる．算術符号につい

ても同様のことが言える．ハフマン符号はブロック長 2 の分節木がもっとも遅延が小さかったが, 符号化に掛かる時間のために無圧縮に比べて全体的に遅延が大きくなった．タンストール符号においてもハフマン符号と同様のことが言えるが, 実験で用いたタンストール分節木は符号語数が 3 のものなので, シンボルが深さが 1 の枝を辿った場合に符号化に掛かる時間が発生しないために, ハフマン符号よりも遅延が小さくなったと考えられる．これらの結果から, 確率 0.5 のときは無圧縮で伝送すべきであると言える．

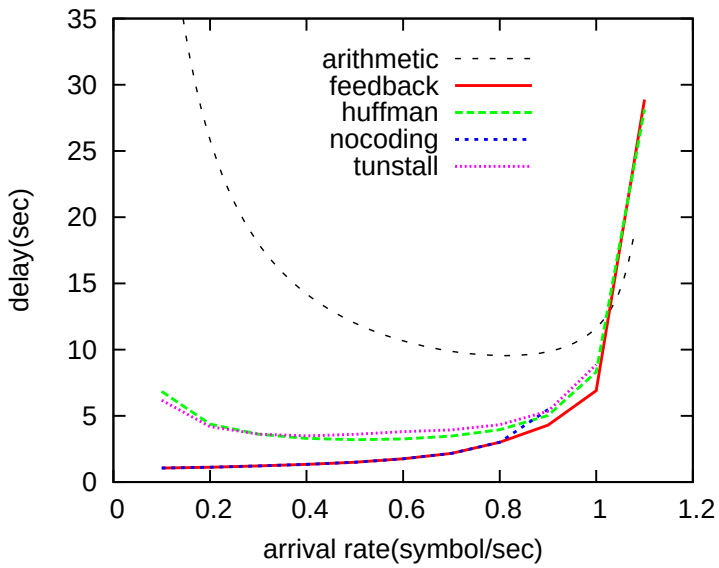

図 8　prob0=0.3 のときの各符号の遅延特性

次に, 確率 0.3 の場合の実験結果について述べる．確率 0.3 の場合の実験結果を図 8 に示す．図 8 を見ると全体的に算術符号の伝送遅延が大きく, ハフマン符号とタンストール符号は到着レートが低い領域ではタンストール符号のほうが遅延が小さく, 到着レートが 0.3 より高い領域ではハフマン符号のほうが遅延が小さいことが分かる．無圧縮と提案法による符号では, 到着レートが 0.8 までは提案法も無圧縮となるが, 到着レートが 0.8 以上の領域では提案法による符号のほうが遅延特性に優れることが分かった．算術符号は, 確率 0.5 の場合と比較して, 確率の偏りが大きくなるほどに符号化に時間が掛かる特性により, 遅延が大きくなった．ハフマン符号は確率 0.3 の場合もブロック長 2 の分節木の遅延が最小であり, 符号化に掛かる時間は確率 0.5 の場合と同じであるが, 確率の偏りにより平均符号語長を短く出来るので遅延の改善が見られた．タンストール符号は, 確率 0.5 の場合と比較して全体的に遅延が増加した, これは用いた符号語数が 3 のタンストール木では確率 0.5 の場合に比べて, 深さが 2 の枝を辿る頻度が多くなるため, 符号化に掛かる時間が増加したためであると考えられる．提案法による符号

は, 到着レートが 0.8 以上の領域では, 分節木の形は符号語数 3 のタンストール木と同じ形をしており, 葉に割り当てられている符号語は需要に応じて最適な長さになっている. これにより, タンストール符号よりも圧縮率が高まり, ハフマン符号より符号化に掛かる時間が短くなるため, 結果的に到着レート 0.8 以上の領域で無圧縮や他の符号よりも遅延が小さくなったと考えられる. 次に, さらに確率の偏りが大きい確率 0.1,0.01 の場合について述べる.

図 9 は確率 0.1 の場合を示しており, 図 10 図 11 は確率 0.01 の場合を示している. まず, 算術符号の遅延特性を図 9 図 10 から見ていくと, 確率 0.5,0.3 の場合同様に他の符号に比べて到着レート全域に渡って遅延が大きくなることが分かった. 算術符号以外の符号については, 図 9 と図 10 の算術符号以外の符号を拡大して示した図 11 より見ていくと, 各符号とも圧縮率が上昇するので, 無圧縮では伝送できない領域でも伝送可能となっていることが分かる. しかし, タンストール符号はシンボルの入力長を長くしなければ圧縮率を高めることができないので, ハフマン符号や提案法による符号と比較すると全体的に遅延が大きいことが分かる. 図 9 図 11 を見ると確率が小さくなるほどに提案法の遅延特性が優れていることが分かる. これは, 分節木の形が到着レートに対応したものになり（言い換えると, 到着レートが高いと深い木になる）, ハフマン符号化により割り当てられる符号語が, ハフマン符号の特性により圧縮率の高いものになるからである. また, 図 11 を見ると, 提案法による符号は他の符号よりも高い到着レートにおいても低い伝送遅延を保ったまま伝送可能となっており, 遅延特性と伝送効率に優れた符号であると言える.

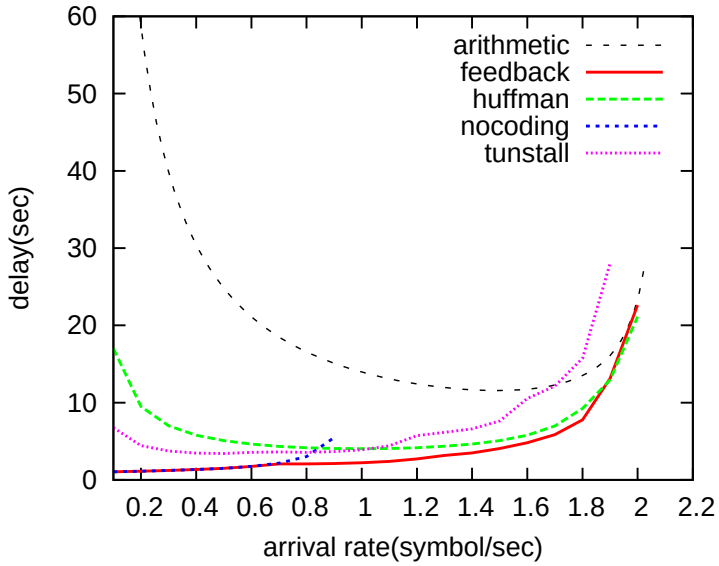

図 9　prob0=0.1 のときの各符号の遅延特性

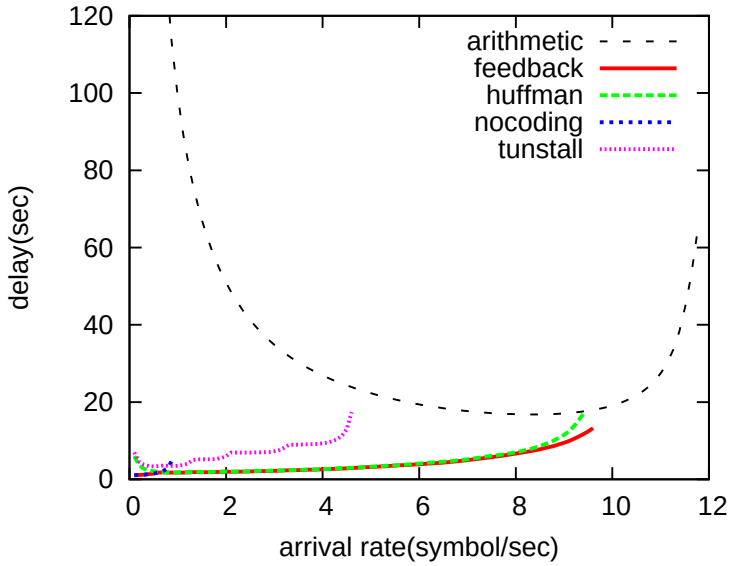

図 10　prob0=0.01 のときの各符号の遅延特性

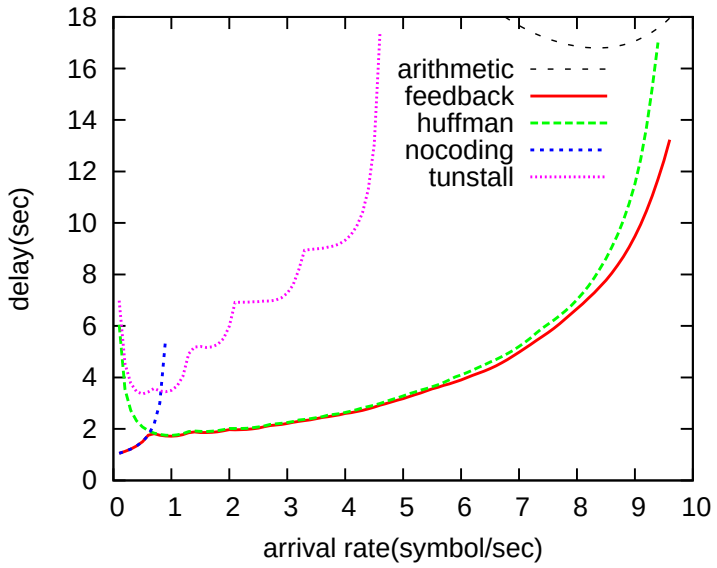

図 11　prob0=0.01 のときの各符号の遅延特性 (拡大)

4.2 伝送レートに対する検討

前節では, 伝送遅延の大きさで各符号を評価してきたが, この節では, 各符号の圧縮率と伝送遅延の両方を評価することにより実際の通信路への応用について考えてゆく. 到着レートの理論限界は, 情報源のエントロピー $H(X)$ から以下の関係を使って求めることが出来る.

$$\frac{\text{伝送レート } (bit/sec)}{\text{到着レート } (symbol/sec)} = \text{符号化レート } (bit/symbol) \geq H(X) \tag{1}$$

本研究では, 伝送レート (bit/sec) は 1(bit/sec) としているので,

$$\frac{1}{\text{到着レート}} \geq H(X) \tag{2}$$

となり, 理論限界は以下の式より得られる.

$$\text{到着レート} \leq \frac{1}{H(X)} \tag{3}$$

これを踏まえて, 到着レートの理論限界が大きくなる確率 0.01 のとき (図 10) について考えると, エントロピー $H(X)$ は

$$H(X) = 0.01 \times \log \frac{1}{0.01} + 0.99 \times \log \frac{1}{0.99} = 0.0808 \tag{4}$$

となり, 到着レートの理論限界は

$$\frac{1}{H(X)} = \frac{1}{0.0808} = 12.3762 \tag{5}$$

となる.

図 10 を見ると算術符号だけが理論限界付近まで伝送可能である. 特に, 到着レートが 12 付近を見ると他の符号では伝送不可能な高い到着レートにおいても, 比較的小さな遅延で伝送可能であることが分かる. そこで, 実際の伝送路で情報源符号化による伝送遅延の改善を考えるときに, ユーザーに予め許容できる遅延を選択して貰い, その上で最適な通信路 (ADSL,FTTH, など) を選択する状況が考えられる. そこで, 伝送レートに対する遅延特性を評価することにより, ユーザーに対してどのようにサービスを提案するべきかを考えた. 評価の際, 各符号の伝送レートに対する遅延特性を用いた.

確率 0.3 と 0.1 のときの伝送レートに対する遅延特性は図 8 図 9 のように算術符号以外は, 遅延に大きな違いは無いことが分かる. 各符号の圧縮率が, さらに高まる確率 0.01 のときの図 14 を見ると提案法による符号とハフマン符号の遅延特性が特に優れていることが分かる. 算術符号やタンストール符号も無圧縮では伝送できない低い伝送レートにおいても伝送可能であるが, 提案法による符号やハフマン符号よりも遅延が大きく. 特に算術符号は, 伝送レート 1

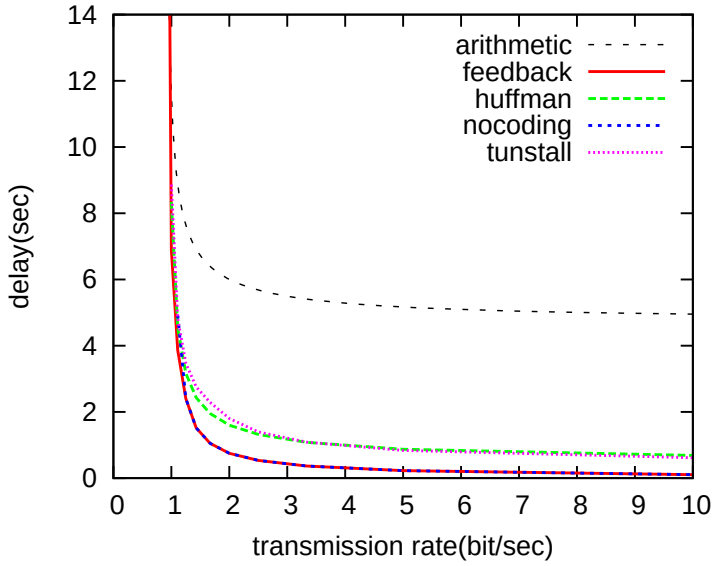

図 12　prob0=0.3 のときの伝送レートに対する遅延特性

のとき他の符号の約 100 倍の遅延となることが分かる (図 14). 以上のことから, 算術符号は高い圧縮率を誇るので, ユーザーが大きい遅延を許容できる場合には, よりコストの低い通信路を提案できるが, 伝送遅延を改善する目的には, 不向きであることが分かった. 以上のことをまとめると, 提案法による符号は, 高い到着レートに対して, 従来用いたどの符号よりも優れた遅延特性を示すことが分かった. しかし, ハフマン符号も提案法に匹敵する遅延特性を示しており, 提案法よりも符号構成が簡単であることなどを考慮すると実用的にはハフマン符号を用いたほうが, 良い場合もあると考えられる.

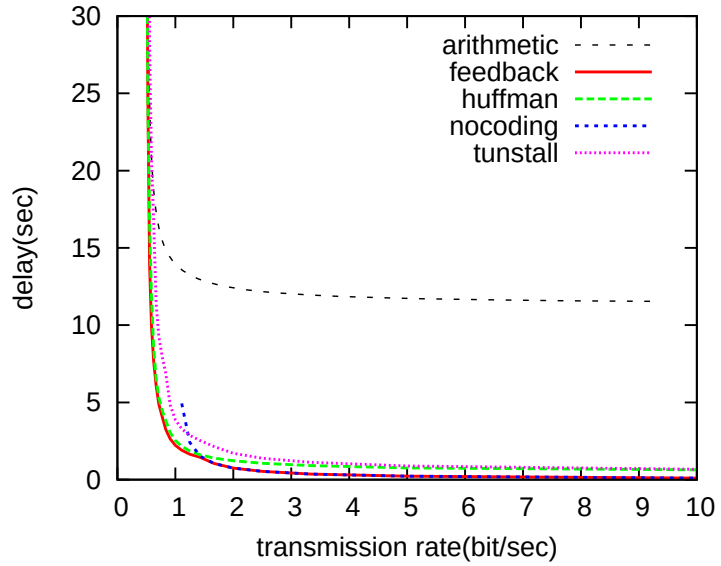

図 13　prob0=0.1 のときの伝送レートに対する遅延特性

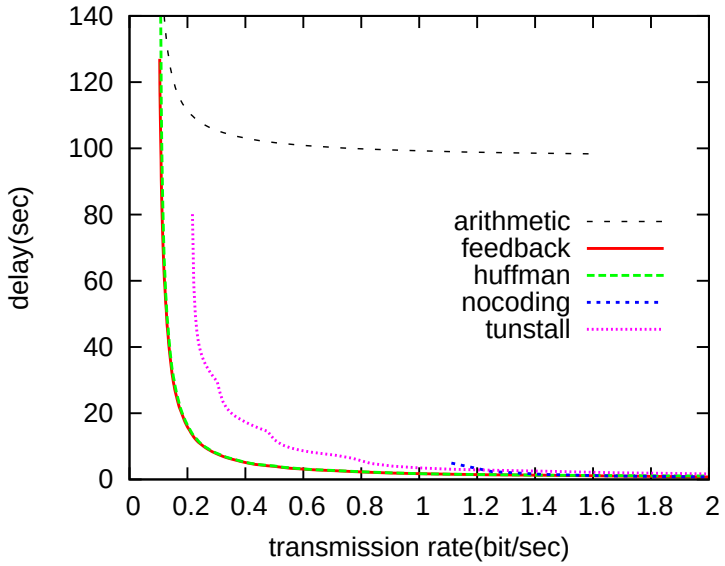

図 14　prob0=0.01 のときの伝送レートに対する遅延特性

5 まとめ

従来研究では, 算術符号とハフマン符号, タンストール符号を用いた伝送遅延の改善が行われていたが, 情報源圧縮を行うことで, 符号化による遅延が発生するため, 情報源シンボルの到着レートが低い領域においては, 無圧縮と比較し遅延が大きくなる傾向があった.

そこで算術符号を用いた伝送システムにおいて, 送信機から送信機の遊休状態を符号器に伝えるフィードバックという機能を加えることで, 符号化に掛かる時間を短くし, 遅延を改善するといったことが, その後の研究で行われていた. しかし, ハフマン符号とタンストール符号については, 到着レートが低い領域で遅延が大きくなることは判明していたが, フィードバックを用いて符号化にかかる時間を短縮するといったことは行われてこなかった. そこで, 本研究の予備実験として, ハフマン符号とタンストール符号にフィードバックを適用したところ, 遅延の改善が見られた. そこで, 本研究では, 分節木の中間ノードに符号語を割り当てることで簡単にフィードバックが可能になるという点に着目し, ハフマン符号やタンストール符号などの既存の符号にこだわらずに, より広い符号の範囲を持つ VV 符号 (variable-length-to-variable-length code) の中から遅延特性に優れる符号を見つけることを考えた. そして, 実験的に求めた需要を用いて, 遅延をより小さくする符号の構成法を提案した. この提案法による符号により, 従来研究で用いてきた符号よりも優れた遅延特性が得られたが, この符号が本研究で設定した枠組みの中で, 伝送遅延を最小にするという意味で最適な物なのかは分からない. また, 符号は実験的に生成するので, 定式的に符号を構成する方法は確立されていない. 従って, 提案法による符号の定式的な構成法を確立することが, 今後の課題である.

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の意を申し上げます.

参考文献

[1] Stephane Musy, Emre Telatar, ”On the transmission of bursty sources,” ISIT 2006, pp.2899-2903, 2006.

[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シンボルの遅延」, 信州大学工学部, 学士論文,2009.

[4] 森田圭一,「ポアソン到着シンボルに対する伝送システムとその遅延特性」, 信州大学大学院工学系研究科, 修士論文, 2010.
[5] 泉 陽一,「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」, 信州大学工学部, 学士論文, 2010.

付録 A　ソースコード

A.1　伝送実験のプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.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_x;
double rand_(void)
{
```

```
    return (-(1/ramda_x)*log(1-(rnd())));/* 指数分布 要修正 */
}
/*****************************************************************************************************/
FILE *fp_source;
FILE *fp_tree;
FILE *fp;
FILE *fp_s;
FILE *fp_c;
FILE *fp_delay;
FILE *fp_demandcount;
FILE *fp_sketch;
FILE *fp_encode_delay;
FILE *fp_sending_delay;
char *tree_code;
int node_index;
int node_sum;
double prob0;
double prob1;
double present_time;/* 現在時刻 */
double arrival_time;/* 到着時刻 */
double sending_time;/* 送信終了時刻 */
double send_time;
#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;
int id_count=0;
/****************************** エンコーダでの遅延計測用 ***********************************************/
double arriving_time[1000];
int end_symbol=0;
/*****************************************************************************************************/
#define INPUT_LENGTH (1 << 20)
int symbol_count;
struct node *inode;/* 構造体 node のポインタ*inode を宣言 */
struct node
{
    struct node *parent;
    struct node *l_ch;
    struct node *r_ch;
    char branch;
    char code_symbol;
    int id;
    int demand_count;
    int code_length;
};
struct node *node;
struct decode
{
    char *code;
    int code_length;
};
struct decode *decode;
int c_node = 0;
struct node *
    input_tree_rec(struct node * current, struct node * parent){
        switch (tree_code[c_node++]){
        case '0':
            current->code_symbol = '0';
            current->l_ch = input_tree_rec(&node[++node_index], current);
            current->r_ch = input_tree_rec(&node[++node_index], current);
            current->parent = parent;
            break;
        case '1':
```

```
            current->code_symbol = '1';
            current->l_ch = NULL;
            current->r_ch = NULL;
            current->parent = parent;
            break;
        default:
            printf("tree code error.\n");
            exit(EXIT_FAILURE);
            break;
        }
        return(current);
}
void make_branch(){
    int i;
    for (i = 0; i < node_sum; i++){
        if (node[i].parent!=NULL){
            if (&node[i] == node[i].parent->l_ch){
                node[i].branch = '0';
            } else {
                node[i].branch = '1';
            }
        }
    }
}
/***************************** ツリーをマッピング ************************************************************/
void read_tree(void)
{
    char tree_data[1000];
    int i;
    int ch;
    int code_length;
    fp_tree = fopen("tree_data.txt","r");
    if (fp_tree == NULL) {
        printf("file open error no tree_data.txt line[%d]\n",__LINE__);
        exit(EXIT_FAILURE);
    }
    for (node_sum = 0; fgets(tree_data,1000,fp_tree) != NULL; node_sum++){
        if (tree_data[0] != '0' && tree_data[0] != '1' && tree_data[0] != '\n'){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
        if (tree_data[0] == '\n')break;
        if (tree_data[1] != ':'){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
        if (strlen(tree_data) < 3){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
    }
    tree_code = (char *)calloc(node_sum,sizeof(char));
    if (tree_code == NULL) {
        printf("error on %d\n",__LINE__);
        exit(EXIT_FAILURE);
    }
    node = (struct node *)calloc(node_sum, sizeof(struct node));/* 必要なノード数分の sutruct node をメモリに
確保 */
    if (node == NULL){
        printf("error on %d\n", __LINE__);
        exit(EXIT_FAILURE);
    }
    decode = (struct decode *)calloc(node_sum,sizeof(struct decode));
    if (tree_code == NULL) {
        printf("error on %d\n",__LINE__);
        exit(EXIT_FAILURE);
    }
    fseek(fp_tree,0,SEEK_SET);
    for (i = 0; fgets(tree_data,1000,fp_tree) != NULL; i++){
```

```
        if (i >= node_sum)break;
        tree_code[i] = tree_data[0];
        for (ch = 0; tree_data[ch] != '\0'; ch++);
        if (tree_data[ch - 1] != '\n'){
            strcpy(tree_data + ch,"\n\0");
        }/* tree_data.txt の最後の行で改行していない場合の対策  \0 -> \n\0 */
        for (code_length = 0; tree_data[code_length] != '\n'; code_length++);
        decode[i].code_length = code_length - 2;
        node[i].code_length = code_length -2;
        node[i].id = i;
        if (tree_data[2] != '\n' && tree_data[2] != '\0'){/* tree_data.txt の 3 列目に符号語があるならば */
            decode[i].code = (char *)calloc(decode[i].code_length + 2,sizeof(char));/* decode[i].code_length に
足されている 2 は"\n\0"の分 */
            if (decode[i].code == NULL) {
                printf("error on %d\n",__LINE__);
                exit(EXIT_FAILURE);
            }
            strcpy(decode[i].code,&tree_data[2]);/* 符号語の列以降にゴミが入る */
        }
    }
    return;
}
/***************************** バッファ(エンコーダ側)*************************************************************/
void add_arriving_list(double time)
{
    if (arriving_count >= ARRIVING_LIST_MAX){
        printf("warning1!!\n");
        exit(EXIT_FAILURE);
    }
    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!!\n");
        exit(EXIT_FAILURE);
    }
    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!!\n");
        exit(EXIT_FAILURE);
    }
    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!!\n");
        exit(EXIT_FAILURE);
    }
    id=sending_buffer_list[sending_buffer_out];
    sending_buffer_out=(sending_buffer_out+1)%SENDING_BUFFER_LIST_MAX;
    sending_buffer_count--;
    return(id);
```

```
}
/***************************** シンボルを生成 ****************************************************************/
int
    generate_symbol()
{
    int x;
    x = (rnd() <= prob0)? '0': '1';
    fprintf(fp_source,"%c\n",x);
    return(x);
}
/***************************** エンコード ********************************************************************/
struct node *node_state;
void
    savesource(struct node *output)/* 符号語を出力する過程で元のシンボルをテキストファイルに出力する */
{
    if (output->parent != NULL){
        savesource(output->parent);
        fprintf(fp_s,"%c\n",output->branch);
    }
    return;
}
void
    encode()/* bit = seq[i] ポインタ (seq[i]) を引数 (bit) として受け取る */
{
    int i;
    char symbol;
    symbol = generate_symbol();
    node_state = (symbol == '0')? node_state->l_ch : node_state->r_ch;
    if (node_state->code_symbol == '1' || (node_state->code_length != 0 && sending_buffer_count == 0)){/* もしノードが葉 ('1') もしくはノードに符号語がありかつ送信機のリングバッファが空なら *//* フィードバック処理 */
        for (i = 0; i < end_symbol; i++){
            fprintf(fp_encode_delay,"%f\n",present_time - arriving_time[i]);
            fprintf(fp_sending_delay,"%d\n",node_state->code_length);
        }
        end_symbol = 0;
        savesource(node_state);/* 情報源シンボルをテキストファイルに出力 */
        if (node_state->code_symbol == '1' && sending_buffer_count != 0){
            node_state->demand_count++;
        }
        add_sending_buffer_list(node_state->id);/* 送信機のリングバッファに符号の番号 (node_state) を保存 */
        if (sending_time < 0.0){
            sending_time = present_time + node_state->code_length;
        }
        node_state = node;/* ポインタ (node_state) をルート (node) に戻す */
    }
    return;
}
int feedback = 0;
int
    feed_back()
{
    int i;
    node_state->demand_count++;/* ノードの需要をカウント 2/2 */
    if (node_state->code_length != 0){
        for (i = 0; i < end_symbol; i++){
            fprintf(fp_encode_delay,"%f\n",present_time - arriving_time[i]);
            fprintf(fp_sending_delay,"%d\n",node_state->code_length);
        }
        end_symbol = 0;
        savesource(node_state);
        add_sending_buffer_list(node_state->id);
    } else {
        return(0);
    }
    return(1);
}
int
    source_length(int id, int s_length){
        if (node[id].parent != NULL){
```

```
            s_length++;
            s_length = source_length(node[id].parent->id,s_length);
        }
        return(s_length);
}
void
    sending_finish()/* 送信終了処理 */
{
    int id;
    int i;
    id = get_sending_buffer_list();
    fprintf(fp_c,"%s",decode[id].code);
    for (i = 0; i < source_length(id,0); i++){
        fprintf(fp_delay,"%f\n",present_time - get_arriving_list());/* delay 表示 */
    }
    if (sending_buffer_count == 0){/* 送信機のバッファに符号語 (id) がないなら */
        if (feed_back()){/* フィードバック */
            sending_time = present_time + node[node_state->id].code_length;
            node_state = node;
        } else {
            sending_time = -1;
        }
    } else {
        id = sending_buffer_list[sending_buffer_out];
        sending_time = present_time + node[id].code_length;
    }
    return;
}
/***************************** 確認用 ************************************************************************/
void
    print_tree_rec(struct node *node)
{
    printf("(");
    printf("%c",node->code_symbol);
    if (node->l_ch != NULL){
        print_tree_rec(node->l_ch);
    }
    if (node->r_ch != NULL){
        print_tree_rec(node->r_ch);
    }
    printf(")");
}
void node_length_rec(struct node *node_state)
{
    if(node_state->l_ch != NULL){
        printf("node_symbol = %c code_length = %d node_id = %d\n",node_state->l_ch->code_symbol,node_state
->l_ch->code_length,node_state->l_ch->id);
        node_length_rec(node_state->l_ch);
    }
    if(node_state->r_ch != NULL){
        printf("node_symbol = %c code_length = %d node_id = %d\n",node_state->r_ch->code_symbol,node_state
->r_ch->code_length,node_state->r_ch->id);
        node_length_rec(node_state->r_ch);
    }
}
/***************************** 到着シンボル処理 ****************************************************************/
void
    arrival_symbol()/* 到着シンボル処理 */
{
    add_arriving_list(present_time);
    arriving_time[end_symbol++] = present_time;
    encode();/* 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;
```

```
}
void
    demand_count()
{
    int i;
    if ((fp_demandcount = fopen("demand_count.txt","w")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < node_sum; i++){
        fprintf(fp_demandcount,"%d\n",node[i].demand_count);
    }
    fclose(fp_demandcount);
}
void
    ini_demand_count()
{
    int i;
    for (i = 0; i < node_sum; i++){
        node[i].demand_count = 0;
    }
}
/***************************** 木を描く ***********************************************************************/
void sketch_tree_rec(struct node *node){
    if (node->code_symbol == '0'){
        if (node->parent != NULL)fprintf(fp_sketch,"\\chunk\n{\n");
        fprintf(fp_sketch,"\\begin{bundle}{%d}\n",node->demand_count);
        sketch_tree_rec(node->l_ch);
        sketch_tree_rec(node->r_ch);
        fprintf(fp_sketch,"\\end{bundle}\n");
        if (node->parent != NULL)fprintf(fp_sketch,"}\n");
    } else {
        fprintf(fp_sketch,"\\chunk{%d}\n",node->demand_count);
    }
}
void sketch_tree()
{
    if ((fp_sketch = fopen("sketch/sketch.tex", "w")) == NULL) {
        system ("mkdir sketch");
        if ((fp_sketch = fopen("sketch/sketch.tex", "w")) == NULL) {
            printf("file open error!!\n");
            exit(EXIT_FAILURE);
        }
    }
    fprintf(fp_sketch,"\\documentclass{jarticle}\n\\usepackage{ecltree,epic,eepic}\n\n\\begin{document}
\n\n\\GapDepth=40pt\n\n");
    sketch_tree_rec(node);
    fprintf(fp_sketch,"\\end{document}\n");
}
int main(int argc,char *argv[])
{
    int i;
    if (argc != 3){
        printf("no ramda_x (0.1〜0.9)\n");
        printf("no prob0\n");
        exit(EXIT_FAILURE);
    }
    ramda_x = atof(argv[1]);
    prob0 = atof(argv[2]);
    init_rnd(1234);
    id_count = 0;
    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;
    if ((fp_s = fopen("source.txt", "wt")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
```

```
    if ((fp_source = fopen("raw_source.txt", "wt")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    if ((fp_c = fopen("code_word.txt", "wt")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    if ((fp_delay = fopen("new_delay.txt", "wt")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    if ((fp_encode_delay = fopen("new_encode_delay.txt", "wt")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    if ((fp_sending_delay = fopen("new_sending_delay.txt", "wt")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    read_tree();/* tree_data.txt より符号化ツリーを読み込む */
    node[0].parent = NULL;
    input_tree_rec(node, NULL);
    if (tree_code[c_node] ==  '0' || tree_code[c_node] == '1'){/* tree が完成しているのに tree_code にシンボ
ルが残っているなら強制終了 */
        printf("tree_data.txt error\n");
        exit(EXIT_FAILURE);
    }
    make_branch();
    node_state = node;
    ini_demand_count();
    /************************ イベントドリブン (feedback Ver.) ************************************/
    while ((arrival_time >= 0.0) || (sending_time >= 0.0)){/* arrival_time と sending_time の両方が 0.0 より
小さい場合処理終了 */
        if (arrival_time < 0 || (sending_time >= 0 && arrival_time > sending_time)){
            present_time = sending_time;/* arrival_time が 0 より小さいか,sending_time が 0 以上でかつ
arrival_time が sending_time よりも大きいの片方もしくは両方ならば,present_time に sending_time を代入 */
            sending_finish();/* 送信終了処理 */
        } else {
            present_time = arrival_time;/* arrival_time が 0 以上でかつ sending_time が 0 より小さい (∞) また
は,arrival_time より大きいならば,present_time に arrival_time を代入する */
            arrival_symbol();/* 到着シンボル処理 */
        }
    }
    /*******************************************************************************************/
    node[0].demand_count = 0;/* ルートの需要は 0 にする */
    demand_count();
    sketch_tree();
    fclose(fp_source);
    fclose(fp_tree);
    fclose(fp_s);
    fclose(fp_c);
    fclose(fp_delay);
    fclose(fp_sending_delay);
    fclose(fp_encode_delay);
    for (i=0; i < node_sum; i++){
        if (decode[i].code_length != 0){
            free(decode[i].code);
        }
    }
    free(decode);
    free(tree_code);
    free(node);
    return 0;
}
```

A.2 付録 A.1 のヘッダファイル

```
#define CODE_VALUE_BITS 16
#define TOP_VALUE (((long)1 << CODE_VALUE_BITS) - 1)
#define FIRST_QTR (TOP_VALUE / 4 + 1)
#define HALF (2 * FIRST_QTR)
#define THIRD_QTR (3 * FIRST_QTR)

#define NO_OF_CHARS (2)
#define MAX_FREQUENCY (16383)
```

A.3 分節木に符号語を割り当てるプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
FILE *read_freq;
FILE *write_code_word;
FILE *read_node;
FILE *n_tree;
FILE *read_nstate;
char r_freq[1000];
char r_node[1000];
char r_nstate[1000];
char tree_data[1000];
char NEW_CODE[1000];
int node_sum;
int new_code_num;
int next_ch;
int bit ;
int threshold;
/***************************** ハフマン符号化 ****************************************************************/
typedef struct t_huff t_huff;
struct t_huff {
    char codesymbol;
    t_huff *parent;
    t_huff *child[2];
};
typedef struct t_list t_list;
struct t_list {
    char new_code[1000];
    char n_state;
    char c_state;
    int id;
    t_huff *p_Leaf;
};
typedef struct t_leaf {
    int entry_num;
    int freq;
    t_huff *node;
} t_leaf;
t_huff *Huffmantree;
t_list *List;
t_leaf *Leaf;
int leaf_cmp(t_leaf *leaf_a, t_leaf *leaf_b)
{
    double diff = leaf_b->freq - leaf_a->freq;
    return((diff > 0)? 1: (diff < 0)? -1: 0);
}
void
    maketree()
```

```
{
    int i;
    int l;
    int smallest;
    int freq;
    t_huff *newnodep;
    if ((read_node = fopen("tree_data.txt", "r")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    if ((read_nstate = fopen("node_state.txt","r")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    if ((read_freq = fopen("demand_count.txt", "r")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    for (node_sum = 0; fgets(tree_data,1000,read_node) != NULL; node_sum++){
        if (tree_data[0] != '0' && tree_data[0] != '1' && tree_data[0] != '\n'){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
        if (tree_data[0] == '\n')break;
        if (tree_data[1] != ':'){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
        if (strlen(tree_data) < 3){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
    }
    fseek(read_node,0,SEEK_SET);
    List = (t_list *)calloc(node_sum, sizeof(t_list));
    if (List == NULL){
        printf("%s in %d\n", "calloc error", __LINE__);
        exit(EXIT_FAILURE);
    }
    Leaf = (t_leaf *)calloc(node_sum, sizeof(t_leaf));
    if (Leaf == NULL){
        printf("%s in %d\n", "calloc error", __LINE__);
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < node_sum; i++){
        fgets(r_node,1000,read_node);
        List[i].n_state = r_node[0];
    }
    for (i = 0; i < node_sum; i++){
        fgets(r_nstate,1000,read_nstate);
        List[i].c_state = r_nstate[0];
    }
    for (i = 0, l = 0; fgets(r_freq,1000,read_freq) != NULL; i++){
        if (List[i].n_state == '1' || (atoi(r_freq) > threshold && List[i].c_state == '1'))l++;
    }
    new_code_num = l;
    fseek(read_freq,0,SEEK_SET);
    Huffmantree = (t_huff *)calloc(((new_code_num * 2) - 1), sizeof(t_huff));
    if (Huffmantree == NULL){
        printf("%s in %d\n", "calloc error", __LINE__);
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < new_code_num; i++){
        Huffmantree[i].child[0] = NULL;
        Huffmantree[i].child[1] = NULL;
    }
    l = 0;
    for (i=0; fgets(r_freq,1000,read_freq) != NULL; i++){/* 葉に需要 (freq) を登録 */
        freq = atoi(r_freq);
```

```
        if (List[i].n_state == '1' || (freq > threshold && List[i].c_state == '1')){
            List[i].p_Leaf = Huffmantree + l;
            Leaf[l].node = Huffmantree + l;
            Leaf[l].freq = freq;
            l++;
        }else{
            List[i].p_Leaf = NULL;
        }
    }
    newnodep = Huffmantree + l;
    qsort(Leaf, l, sizeof(t_leaf), (int(*)(const void *, const void *))leaf_cmp);/* Leaf を freq が大きい順
に並べ替え */
    for (smallest = l - 1; smallest > 0; smallest--, newnodep++){
        Leaf[smallest - 1].node->codesymbol = '0';
        Leaf[smallest    ].node->codesymbol = '1';
        newnodep->child[0] = Leaf[smallest - 1].node;
        newnodep->child[1] = Leaf[smallest    ].node;
        Leaf[smallest - 1].node->parent = newnodep;
        Leaf[smallest    ].node->parent = newnodep;
        Leaf[smallest - 1].node = newnodep;
        Leaf[smallest - 1].freq += Leaf[smallest].freq;
        for (i = smallest - 1; i > 0 && Leaf[i].freq > Leaf[i - 1].freq; i--){/* ハフマン符号化の過程におけ
る葉の入れ替え */
            t_leaf tmp;
            tmp = Leaf[i];
            Leaf[i] = Leaf[i - 1];
            Leaf[i - 1] = tmp;
        }
    }
    Leaf[0].node->parent = NULL;
    fclose(read_node);
    fclose(read_freq);
    return;
}
int
    make_code_rec(t_huff *node,int count){
        if (node->parent != NULL){
            count = make_code_rec(node->parent,count);
            count++;
        }
        NEW_CODE[count] = node->codesymbol;
        NEW_CODE[count + 1] = '\n';
        return(count);
}
void
    new_tree(){
        int i;
        int l;
        int a;
        l = 0;
        for (i = 0; i < node_sum; i++){
            if (List[i].n_state == '1' || List[i].p_Leaf != NULL){
                make_code_rec(Huffmantree + l,-1);
                l++;
                for (a = 0; NEW_CODE[a] != '\n'; a++){
                    List[i].new_code[a] = NEW_CODE[a];
                }
            }
        }
}
void
    out_tree(){
        int i;
        if ((n_tree = fopen("tree_data_new.txt", "wt")) == NULL) {
            printf("file open error!!\n");
            exit(EXIT_FAILURE);
        }
        for (i = 0; i < node_sum; i++){
            fprintf(n_tree,"%c:%s\n",List[i].n_state,List[i].new_code);
```

```
        }
        fclose(n_tree);
        fclose(read_nstate);
        return;
}
/*****************************************************************************************************/
int
    main(int argc,char *argv[])
{
    if (argc == 1){
        threshold = 0;
    } else if (argc == 2 && 0 <= atoi(argv[1]) && atoi(argv[1]) <= 1000000){
        threshold = atoi(argv[1]);/* 中間ノードに符号語の割り当てを行わない，需要の上限値（未入力では需要 0 のとき
符号語を割り当てない）*/
    } else {
        printf("Threshold error\n");
        exit(EXIT_FAILURE);
    }
    maketree();/* tree_data.txt より符号化ツリーを読み込む */
    new_tree();/* ハフマン符号化により符号語を割り当て直された新しいツリーをプログラム上に作成 */
    out_tree();/* 新しいツリーを tree_data_new.txt に出力する */
    free(List);
    free(Leaf);
    free(Huffmantree);
    return (0);
}
```

A.4 分節木を変形するプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
FILE *read_freq;
FILE *write_code_word;
FILE *read_node;
FILE *n_tree;
char r_freq[1000];
char r_node[1000];
char tree_data[1000];
char NEW_CODE[1000];
int node_sum;
int new_code_num;
int next_ch;
int bit ;
int threshold;
/***************************** ハフマン符号化 ***************************************************************/
typedef struct t_huff t_huff;
struct t_huff {
    char codesymbol;
    t_huff *parent;
    t_huff *child[2];
};
typedef struct t_list t_list;
struct t_list {
    char new_code[1000];
    char c_state;
    int id;
    t_huff *p_Leaf;
};
typedef struct t_leaf {
    int entry_num;
    int freq;
    t_huff *node;
} t_leaf;
```

```
t_huff *Huffmantree;
t_list *List;
t_leaf *Leaf;
int leaf_cmp(t_leaf *leaf_a, t_leaf *leaf_b)
{
    double diff = leaf_b->freq - leaf_a->freq;
    return((diff > 0)? 1: (diff < 0)? -1: 0);
}
void
    maketree()
{
    int i;
    int l;
    int smallest;
    int freq;
    t_huff *newnodep;
    if ((read_node = fopen("tree_data.txt", "rt")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    if ((read_freq = fopen("demand_count.txt", "rt")) == NULL) {
        printf("file open error!!\n");
        exit(EXIT_FAILURE);
    }
    for (node_sum = 0; fgets(tree_data,1000,read_node) != NULL; node_sum++){
        if (tree_data[0] != '0' && tree_data[0] != '1' && tree_data[0] != '\n'){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
        if (tree_data[0] == '\n')break;
        if (tree_data[1] != ':'){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
        if (strlen(tree_data) < 3){
            printf("tree_data.txt error\n");
            exit(EXIT_FAILURE);
        }
    }
    fseek(read_node,0,SEEK_SET);
    List = (t_list *)calloc(node_sum, sizeof(t_list));
    if (List == NULL){
        printf("%s in %d\n", "calloc error", __LINE__);
        exit(EXIT_FAILURE);
    }
    Leaf = (t_leaf *)calloc(node_sum, sizeof(t_leaf));
    if (Leaf == NULL){
        printf("%s in %d\n", "calloc error", __LINE__);
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < node_sum; i++){
        fgets(r_node,1000,read_node);
        List[i].c_state = r_node[0];
    }
    l = 0;
    for (i = 0; fgets(r_freq,1000,read_freq) != NULL; i++){
        if (List[i].c_state == '1' || atoi(r_freq) > threshold)
            l++;
    }
    new_code_num = l;
    fseek(read_freq,0,SEEK_SET);
    Huffmantree = (t_huff *)calloc(((new_code_num * 2) - 1), sizeof(t_huff));
    if (Huffmantree == NULL){
        printf("%s in %d\n", "calloc error", __LINE__);
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < new_code_num; i++){
        Huffmantree[i].child[0] = NULL;
        Huffmantree[i].child[1] = NULL;
```

```
    }
    l = 0;
    for (i=0; fgets(r_freq,1000,read_freq) != NULL; i++){/* 葉に需要（freq）を登録 */
        freq = atoi(r_freq);
        if (List[i].c_state == '1' || freq > threshold){
            List[i].p_Leaf = Huffmantree + l;
            Leaf[l].node = Huffmantree + l;
            Leaf[l].freq = freq;
            l++;
        }else{
            List[i].p_Leaf = NULL;
        }
    }
    newnodep = Huffmantree + l;
    qsort(Leaf, l, sizeof(t_leaf), (int(*)(const void *, const void *))leaf_cmp);/* Leaf を freq が大きい順
に並べ替え */
    for (smallest = l - 1; smallest > 0; smallest--, newnodep++){
        Leaf[smallest - 1].node->codesymbol = '0';
        Leaf[smallest    ].node->codesymbol = '1';
        newnodep->child[0] = Leaf[smallest - 1].node;
        newnodep->child[1] = Leaf[smallest    ].node;
        Leaf[smallest - 1].node->parent = newnodep;
        Leaf[smallest    ].node->parent = newnodep;
        Leaf[smallest - 1].node = newnodep;
        Leaf[smallest - 1].freq += Leaf[smallest].freq;
        for (i = smallest - 1; i > 0 && Leaf[i].freq > Leaf[i - 1].freq; i--){/* 葉の入れ替え */
            t_leaf tmp;
            tmp = Leaf[i];
            Leaf[i] = Leaf[i - 1];
            Leaf[i - 1] = tmp;
        }
    }
    Leaf[0].node->parent = NULL;
    fclose(read_node);
    fclose(read_freq);
    return;
}
int
    make_code_rec(t_huff *node,int count){
        if (node->parent != NULL){
            count = make_code_rec(node->parent,count);
            count++;
        }
        NEW_CODE[count] = node->codesymbol;
        NEW_CODE[count + 1] = '\n';
        return(count);
}
void
    new_tree(){
        int i;
        int l;
        int a;
        l = 0;
        for (i = 0; i < node_sum; i++){
            if (List[i].c_state =='1' || List[i].p_Leaf != NULL){
                make_code_rec(Huffmantree + l,-1);
                l++;
                for (a=0; NEW_CODE[a] != '\n'; a++){
                    List[i].new_code[a] = NEW_CODE[a];
                }
            }
        }
}
void
    out_tree(){
        int i;
        if ((n_tree = fopen("tree_data_new.txt", "wt")) == NULL) {
            printf("file open error!!\n");
            exit(EXIT_FAILURE);
```

```
        }
        for (i=0; i < node_sum; i++){
            fprintf(n_tree,"%c:%s\n",List[i].c_state,List[i].new_code);
        }
        fclose(n_tree);
}
/*****************************************************************************************************/
int
    main(int argc,char *argv[])
{
    if (argc == 1){
        threshold = 0;
    } else if (argc == 2 && 0 <= atoi(argv[1]) && atoi(argv[1]) <= 1000000){
        threshold = atoi(argv[1]);
    } else {
        printf("Threshold error\n");
        exit(EXIT_FAILURE);
    }
    maketree();/* 外部テキストからツリーと需要を読み込み，ハフマンツリーを作る */
    new_tree();/* ハフマンツリーにより作られた符号語を元のツリーに割り当てる */
    out_tree();/* 符号語が割り当てなおされたツリーを出力する */
    free(List);
    free(Leaf);
    free(Huffmantree);
    return (0);
}
```